Graph Representation Learning for Classification and Anomaly Detection

Mahsa Mesgaran Ayaghchi

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
for the Ph.D. Degree in Information and Systems Engineering at
Concordia University
Montreal, QC, Canada

December 2023

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Mahsa Mesgaran Ayaghchi

Entitled: Graph Representation Learning for Classification and Anomaly Detection

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy Information and Systems Engineering

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Chair
Govind Gopakumar

External Examiner
Wassim Bouachir

Arm's Length Examiner
Farjad Shadmehri

Examiner
Nizar Bouguila

Examiner
Yong Zeng

Thesis Supervisor (s)
Abdessamad Ben Hamza

Approved by

Chun Wang Chair of Department or Graduate Program Director

December 19, 2023
Date of Defence

Mourad Debbabi Dean, Gina Cody School

Abstract

Graph Representation Learning for Classification and Anomaly Detection

Mahsa Mesgaran Ayaghchi, Ph.D

Concordia University, 2023

Graph-structured data is ubiquitous across diverse domains, including social networks, recommendation systems, brain networks, computational chemistry, biology, sensor networks, and transportation networks. Graph neural networks have recently emerged as a powerful paradigm for the analysis of graph-structured data due to their ability to effectively capture complex relationships and learn expressive graph node representations through iterative aggregation of information from neighboring nodes. These learned representations can then be used in various downstream tasks such as node classification and anomaly detection.

In this thesis, we introduce a graph representation learning model for semi-supervised node classification. The proposed feature-preserving model addresses the challenges of oversmoothing and shrinking effects by introducing a nonlinear smoothness term into the feature diffusion mechanism of graph convolutional networks. We conduct comprehensive experiments on diverse benchmark datasets demonstrating that our approach consistently outperforms or matches state-of-the-art baseline methods. Inspired by the concept of implicit fairing in geometry processing, we also propose a graph fairing convolutional network architecture for semi-supervised anomaly detection. The proposed model leverages a feature propagation rule derived directly from the Jacobi iterative method and incorporates skip connections between initial node features and each hidden layer, facilitating robust information propagation throughout the network. Our extensive experiments on five benchmark datasets showcase the superior performance of our graph fairing convolutional network compared to existing anomaly detection methods. In addition, we propose an unsupervised anomaly detection approach on graph-structured data by designing a graph encoder-decoder architecture and a locality-constrained pooling strategy. This pooling mechanism extracts local patterns

and reduces the impact of irrelevant global graph information, enhancing the discriminative power of the learned features. In the decoding phase, an unpooling operation followed by a graph deconvolutional network reconstructs the graph data. Extensive experiments on six benchmark datasets demonstrate that our graph encoder-decoder model outperforms competitive baseline methods.

Acknowledgments

I would like to express my deepest gratitude to all those who have contributed to the successful completion of this PhD thesis. This journey has been a long and challenging one, and I could not have reached this milestone without the support and assistance of many individuals and institutions.

First and foremost, I am indebted to my advisor, Prof. A. Ben Hamza, whose wisdom, patience, and expertise have been invaluable throughout this research journey. His mentorship has not only shaped my academic growth but also my character. His dedication to excellence and tireless commitment to pushing the boundaries of knowledge have been a constant source of inspiration.

I extend my heartfelt thanks to the members of my doctoral thesis examining committee for their critical feedback, insightful suggestions, and dedication to ensuring the rigor and quality of my work.

I am grateful to Concordia University for providing the resources and facilities necessary for conducting my research. The academic personnel and administrative staff, IT support team, and library staff have all played a significant role in facilitating my academic journey.

My family and friends deserve my heartfelt appreciation for their unwavering support, love, and encouragement throughout this academic journey.

Last but not least, I would like to dedicate this thesis to Vahid, my husband, whose love and support have been my anchor during this demanding period of academic pursuit. His sacrifices and belief in me have been my strongest motivators.

In conclusion, this thesis represents the culmination of years of hard work, dedication, and the collective efforts of many individuals. I am profoundly grateful to all of you for being a part of this remarkable journey.

Table of Contents

List of Tables

List of Figures

List of Acronyms

GNN Graph Neural Network

GCN Graph Convolutional Network

LCPool locality-constrained Pooling

SVDD Support Vector Data Description

APPNP Approximate Personalized Propagation of Neural Predictions

ResGCN Residual Graph Neural Network

GCNII Graph Convolutional Network with Initial residual and Identity mapping

DiffPool Differentiable Pooling

SAGPool Self-Attention Graph Pooling

ReLU Rectified Linear Unit

CNN Convolutional Neural Network

ChebyNet Chebyshev Network

AGCN Anisotropic Graph Convolutional Network

GAT Graph Attention Network

JK-Net Jumping Knowledge Network

GFCN Graph Fairing Convolutional Network

LCUnpool locality-constrained Unpooling

LLC Locality-constrained Linear Coding

GRAND Graph Random Neural Networks

ANOMALOUS Anomaly Detection on Attributed Networks

AAGNN Abnormality-Aware Graph Neural Network

CHAPTER 1

Introduction

In this chapter, we start with the motivation behind this work, followed by the problem statement, objectives of the study, and literature review. Then, we present the basic preliminaries and background material, which include a brief overview of graphs, graph embedding, graph embedding applications, graph neural networks, and graph convolutional networks, and finally we conclude with the thesis contributions.

1.1 Framework and Motivation

In the realm of graph-based data representation, graph embedding has become a powerful technique for representing and analyzing graph data. The rapid advancement of deep learning has generated significant interest in the adoption of graph neural networks (GNNs) for the acquisition of latent representations of graphs. One of the pioneering developments in this arena is graph convolutional networks (GCNs) [1]. GCNs have played a central role in this pursuit by utilizing convolutional operations on graphs to learn effective node embeddings that have proven to be useful in achieving high accuracy prediction results. However, GCNs suffer from oversmoothing, where the neighborhood information becomes indistinguishable across layers, and the challenge of capturing long-range dependencies. To address these challenges, we propose a framework, enabling nodes to adaptively aggregate information from their neighbors, thus mitigating oversmoothing while capturing richer local graph structures.

Graph data is ubiquitous in diverse domains, from social networks to recommendation systems, and accurately identifying anomalies within these intricate structures is paramount. Capturing

abnormal nodes in graph-structure dataset is challenging, primarily because anomalies are rare occurrences and only a very small proportion of the graph nodes might be anomalous. To address this fundamental issue, semi-supervised learning techniques are proposed for graph-based anomaly detection. Inspired by the principles of implicit fairing in geometry processing [2] for triangular mesh smoothing, we present a novel semi-supervised framework for graph anomaly detection. Furthermore, we introduce a learnable skip connection mechanism within our model allowing our model to seamlessly integrate initial node representations with aggregated neighborhood information. This amalgamation not only enhances the learning of node representations but also facilitates consistent information flow across network layers, ultimately improving anomaly detection performance.

A substantial challenge in anomaly detection arises from the scarcity of labeled data, as manual annotation of anomalies is both costly and often unfeasible at scale. More recently, unsupervised GNN-based methods have been shown effective at addressing anomaly detection problems in various data settings. A key component of many GNNs is the pooling operation, which seeks to reduce the size of a graph while preserving its vital structural information, crucial for detecting abnormal nodes [3]. Many existing graph pooling techniques rely on complex, trainable parameters, resulting in computational complexity and limited interpretability. We propose an unsupervised graph encoder-decoder model that incorporates a novel pooling mechanism.

1.2 Problem Statement

In this thesis, we briefly describe the semi-supervised node classification, semi-supervised graph anomaly detection, and unsupervised graph anomaly detection.

1.2.1 Semi-Supervised Node Classification

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of N nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. Learning latent representations of nodes in a graph aims at encoding the graph structure into low-dimensional embeddings, such that both structural and semantic information are captured. More precisely, the purpose of network/graph embedding is to learn a mapping $\varphi : \mathcal{V} \to \mathbb{R}^P$ that maps each node i to a P-dimensional vector $\mathbf{z}_i$, where $P \ll N$. These learned node embeddings can then be used as input to learning algorithms for downstream tasks, such as node classification.

Given the labels of a subset of the graph nodes, the objective of semi-supervised learning is to predict the unknown labels of the other nodes. Specifically, let $\mathcal{D}_K = \{(\mathbf{z}_i, y_i)\}_{i=1}^K$ be the set of labeled final output node embeddings $\mathbf{z}_i \in \mathbb{R}^P$ with associated known labels $y_i \in \mathcal{Y}_K$, and

$\mathcal{D}_U = \{\mathbf{z}_i\}_{i=K+1}^{K+U}$ be the set of unlabeled final output node embeddings, where $K+U=N$. Then, the problem of semi-supervised node classification is to learn a classifier $f : \mathcal{V} \to \mathcal{Y}_K$. That is, the goal is to predict the labels of the set $\mathcal{D}_U$.

1.2.2 Semi-Supervised Graph Anomaly Detection

Anomaly detection aims at identifying anomalous instances, which do not conform to the expected pattern of other instances in a dataset. It differs from binary classification in that it distinguishes between normal and anomalous observations. Furthermore, in anomaly detection, the underlying distribution of anomalies is typically not known in advance.

Let $\mathcal{D}_l = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_l}$ be a set of labeled data points $\mathbf{x}_i \in \mathbb{R}^F$ and their associated known labels $y_i \in \{0, 1\}$ with 0 and 1 representing "normal" and "anomalous" observations, respectively, and $\mathcal{D}_u = \{\mathbf{x}_i\}_{i=N_l+1}^{N_l+N_u}$ be a set of unlabeled data points, where $N_l + N_u = N$. Hence, each node i can be labeled with a 2-dimensional one-hot encoding vector $\mathbf{y}_i = (y_i, 1 - y_i)$.

The goal of semi-supervised anomaly detection on graphs is to estimate the anomaly scores of the unlabeled graph nodes. Nodes with elevated anomaly scores are classified as anomalous, whereas those with lower scores are categorized as normal.

1.2.3 Unsupervised Graph Anomaly Detection

Unsupervised node anomaly detection in attributed graphs addresses the challenge of identifying anomalous nodes within a graph without relying on labeled training data or ground truth information. In this context, an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ is given, where $\mathcal{V}$ represents the set of nodes, $\mathcal{E}$ denotes the set of edges, and $\mathbf{X}$ encompasses the attributes associated with each node. The fundamental goal of unsupervised node anomaly detection is to construct a scoring function $s : \mathcal{V} \to \mathbb{R}$ that assigns a numerical anomaly score to each node in the graph.

The scoring function s maps each node in $\mathcal{V}$ to a real-valued anomaly score. This function captures the inherent characteristics and relationships within the graph, allowing for the quantification of node abnormality. Following the computation of anomaly scores for all nodes, a user-defined parameter r is utilized to select the top r nodes with the highest anomaly scores. These selected nodes are subsequently classified as anomalies, signifying their departure from the expected graph behavior.

1.3 Objectives

In this thesis, we aim to achieve the following objectives:

- We present an anisotropic graph convolutional network [4] designed for semi-supervised node classification, with the primary goal of mitigating oversmoothing and shrinking effects. Our proposed framework is constructed as a nonlinear function, capable of capturing valuable node features while preventing oversmoothing. This approach draws inspiration from anisotropic diffusion techniques in image and geometry processing, learning nonlinear representations based on local graph structure and node features.

- We design a novel graph convolutional network with skip connections for semi-supervised anomaly detection on graph-structured data [5] with the aim of improving the flow of information and preventing vanishing gradients. The model is based on the concept of implicit fairing in geometry processing and uses a propagation rule derived from the Jacobi method, combining a graph convolution module for local information aggregation and a skip connection module for merging neighborhood representations across network layers.

- We propose an unsupervised graph encoder-decoder model [6] to enhance unsupervised anomaly detection in attributed graphs with the incorporation of LCPool, a locality-constrained linear coding-based pooling strategy. This approach aims to improve the accuracy and effectiveness of anomaly detection by learning discriminative representations of graph-structured data, emphasizing local patterns, and reducing the impact of noise during encoding and decoding.

1.4 Literature Review

Semi-Supervised Node Classification. In recent years, the growing prevalence of graph-structured data in real-world applications has led to a surge of interest in developing efficient methods for representing graphs. Network embedding has emerged as a powerful approach for representing and analyzing such data [7–10]. The core idea is to learn low-dimensional embedding vectors that capture both structural and semantic information within the graph. These embeddings serve as input for various machine learning tasks, including link prediction, visualization, recommendation, anomaly detection, and node classification.

The literature on network embedding has largely centered on the use of random walks and neural language models to learn effective node embeddings [11–13]. For instance, Perozzi et al. introduced DeepWalk [11], a framework that leverages local information obtained from truncated random walks to learn latent representations of graph nodes. DeepWalk treats each random walk as a sentence, applying the skip-gram language model [14] to maximize co-occurrence probabilities among words within a sentence. Another popular method, node2vec [12], extends DeepWalk by

using second-order random walks with breadth-first and depth-first sampling strategies. However, node2vec requires fine-tuning for different datasets and tasks due to its numerous parameters.

In recent years, the advent of deep learning has led to increased interest in using graph neural networks (GNNs) to learn latent representations of graphs [15, 16]. Many GNNs, inspired by convolutional neural networks (CNNs) and network embedding, have been proposed. For instance, Defferrard et al. introduced Chebyshev networks (ChebyNet) [17], which use recursive Chebyshev polynomial spectral filters to efficiently perform graph convolutions without explicitly computing Laplacian eigenvectors. Graph convolutional networks (GCNs) [1], on the other hand, have become a popular semi-supervised learning framework for graph-based deep learning, utilizing a first-order approximation of spectral graph convolutions. GCN uses an efficient layer-wise propagation rule, which is based on a first-order approximation of spectral graph convolutions. The feature vector of each graph node is updated by essentially applying a weighted sum of the features of its neighboring nodes. Monti *et al.* [18] present a mixture of networks (MoNet) model, a GCN-based model that employs a mixture of Gaussian kernels with learnable parameters to model the weight function of pseudo-coordinates, which are associated to the neighboring nodes of each graph node. Liao *et al.* [19] propose the Lanczos network (LanczosNet), which employs the Lanczos algorithm to construct low-rank approximations of the graph Laplacian in order to facilitate efficient computations of matrix powers. Xu *et al.* [20] introduce a graph wavelet neural network, which is a GCN-based architecture that uses spectral graph wavelets in lieu of graph Fourier bases to define a graph convolution.

While GCNs have shown great promise, achieving state-of-the-art performance on semi-supervised node classification, they are prone to oversmoothing the node features. Wu *et al.* [21] introduce a simple graph convolution by eliminating the nonlinear transition functions between graph convolutional network layers. These functions collapse the resulting function into a single linear transformation via the powers of the normalized adjacency matrix with added self-loops for all graph nodes. However, this simplistic graph convolution primarily functions as a low-pass filter, which attenuates all frequencies except the zero frequency, ultimately leading to the problem of oversmoothing. Recently, significant strides have been made toward remedying the issue of oversmoothing in GCNs [22, 23]. Xu *et al.* [22] propose jumping knowledge networks, which employ dense skip connections to connect each layer of the network with the last layer to preserve the locality of node representations in order circumvent oversmoothing. More recently, Zhao *et al.* [23] proposed a normalization layer, which helps avoid oversmoothing by preventing learned representations of distant nodes from becoming indistinguishable. This normalization layer is performed on intermediate layers during training. The objective is to apply smoothing over nodes within the

same cluster while avoiding smoothing over nodes from different clusters.

Semi-Supervised Graph Anomaly Detection. Anomaly detection aims to identify instances that deviate from the expected pattern of other instances in a dataset. While shallow methods such as one-class classification models [24] require explicit hand-crafted features, many recent advancements in anomaly detection rely on deep learning techniques [25]. These deep learning approaches can automatically extract relevant features from the data without the need for explicit hand-crafted feature engineering. Ruff *et al.* [26] develop an extension of the shallow one-class classification approach [24], known as deep SVDD. Deep SVDD is an unsupervised learning model that learns to extract the common factors of variation of the data distribution. This is achieved by training the neural network to minimize the volume of a hypersphere that encloses the representations of the data generated by the network. Typically, the centroid of this hypersphere is set to the mean of the feature representations, which are learned through a single initial forward pass of the data. In order to improve model performance, Ruff *et al.* [27] propose Deep SAD, a generalization of the unsupervised Deep SVDD to the semi-supervised setting. The key difference between these two deep anomaly detection models is the objective function. However, both Deep SVDD and Deep SAD suffer from the hypersphere collapse problem, which arises from the risk of learning a trivial solution. Specifically, without any constraints on the architecture of the models, the learned features within the neural network have a tendency to converge to the centroid of the hypersphere.

Recently, GCNs have emerged as the predominant semi-supervised model for generating representations from graph data, demonstrating superior performance across various application domains such as anomaly detection [28]. Kumagai *et al.* [29] introduced two semi-supervised anomaly detection models. The first model relies on labeled normal instances, while the second model utilizes both labeled normal and anomalous instances. However, both models are trained to minimize the volume of a hypersphere that encloses the GCN-learned node embeddings of normal instances, and hence they also suffer from the hypersphere collapse problem.

While GCN-based models have demonstrated considerable success in learning node representations from graphs, they are prone to over-smoothing. Recently, several methods have been developed that utilize skip connections to tackle the problem of over-smoothing. JK-Net [22] uses jumping knowledge network connections to connect each layer to the last one, maintaining the feature mappings in lower layers. APPNP [30], which approximate PageRank with power iteration, uses initial connection by connecting each layer to the original feature matrix. By decoupling feature transformation and propagation, APPNP can aggregate information from multi-hop neighbors without increasing the number of layers in the network. ResGCN [31] is a residual graph convolutional network that extends the depth of GCNs by using residual/dense connections and

dilated convolutions. GCNII [32] employs initial residual and identity mapping to mitigate the over-smoothing problem. At each layer, the initial residual constructs a skip connection from the input layer, while the identity mapping adds an identity matrix to the weight matrix.

Unsupervised Graph Anomaly Detection One common approach in graph anomaly detection is to use unsupervised methods, aiming to detect anomalous nodes within a graph without relying on labeled data. This models can be particularly challenging due to the absence of ground truth for what constitutes an anomaly. Ding *et al.* [28] present a GCN-based autoencoder for anomaly detection including three key components: an encoder designed to capture both network structure and node features in order to facilitate node embedding representation learning with GCN, a structure reconstruction decoder that reconstructs the original graph structure using the learned node embeddings, and an attribute reconstruction decoder to reconstruct the observed nodal attributes based on the obtained node embeddings. Wang *et al.* [33] design a graph anomaly detection model based on one-class classification method by mapping the training nodes into a hypersphere in the embedding space via graph neural networks. Zhou *et al.* [34] introduce an abnormality-aware graph neural network. This method employs a subtractive aggregation technique to characterize each node by based on its deviation from its neighboring nodes. Specifically, nodes that are classified as normal with a high degree of confidence are employed as training labels to instruct the network in acquiring a specialized hypersphere criterion. This criterion is then utilized to identify anomalies within the attributed graph for identifying anomalies. Pei *et al.* [35] introduce a GCN-based model that captures the sparsity and nonlinearity present in attributed graphs. This model incorporates residual information and applies a specialized residual-based attention mechanism helping mitigate the adverse effects caused by anomalous nodes in the graph. Zhuang *et al.* [36] propose a subgraph centralization approach addressing the weaknesses of existing detectors in terms of computational cost, suboptimal detection accuracy, and lack of explanation for identified anomalies. This technique demonstrates its ability to identify anomalies in large-scale networks while also offering explanatory insights into the reasons behind a node's anomalous or normal status. Duan *et al.* [37] present a multi-view, multi-scale contrastive learning framework with subgraph-subgraph contrast for graph anomaly detection by combining various anomalous information and calculating the anomaly score for each node.

Graphs can often be very large and complex, making it challenging to identify anomalies. To address this problem, graph pooling is a commonly-used operation in GNNs, with the aim of producing a compact yet informative representation of the graph structure by summarizing the information contained in the nodes of the graph. By applying a pooling operation, the graph can be transformed into a coarse representation that is easier to analyze or use as input for downstream

tasks such as graph anomaly detection. Graph pooling methods can be broadly categorized into two types: global pooling and hierarchical pooling. Global pooling methods summarize the information of all nodes in the graph into a single vector or scalar [38–40], while hierarchical pooling methods recursively apply a pooling operation to the graph, producing a hierarchy of coarser graphs with decreasing numbers of nodes [41–43]. On the other hand, spectral clustering pooling techniques consider graph pooling as a cluster assignment task [43], which categorizes nodes into a set of clusters based on their learned embeddings and constructs the coarser graph based on new nodes using a learned or predefined cluster assignment matrix. Ying *et al.* [41] introduced DiffPool, a differentiable pooling technique aimed at hierarchically representing graphs by learning a cluster assignment matrix in an end-to-end manner. This matrix encapsulates the probabilities of nodes in each layer being assigned to clusters in the next layer, relying on both node attributes and graph topology. Other hierarchical pooling methods include SAGPool [42] and gPool [44]. These methods focus on learning hierarchical graph representations by incorporating both node attributes and graph structure. SAGPool identifies the most crucial nodes based on their self-attention scores. The selected nodes are then retained in the pooled representation, while the remaining nodes are discarded. On the other handgPool utilizes scalar projection values computed with the help of a trainable projection vector to sample nodes, ultimately producing a coarser version of the graph.

1.5 Preliminaries

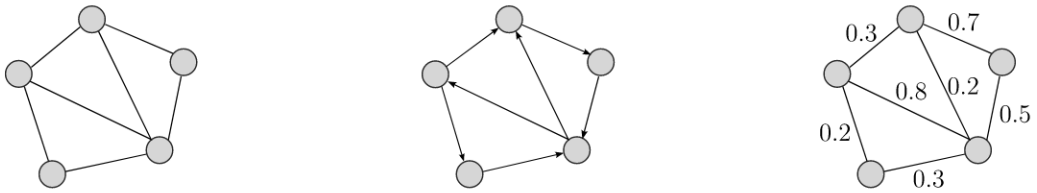
In this section, we present a terse overview of graph theory basics, graph embedding, graph embedding applications, graph neural networks, and graph convolutional networks.

1.5.1 Graph Theory Basics

Graphs serve as powerful tools for representing relationships between entities in various domains, such as social networks, e-commerce platforms, citation networks, geometry processing [45–49], mesh denoising [50], and mesh watermarking [51,52]. Formally, a graph is denoted by $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, \ldots, N\}$ is the set of N nodes or vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges or links connecting pairs of vertices. In Figure 1.1, we depict various types of graphs, including undirected, directed, and weighted graphs, to provide a visual illustration of these definitions.

Graphs can be characterized by their adjacency matrices, degree matrices, and Laplacian matrices, defined as follows:

Adjacency matrix An adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ encodes the relationships between nodes in a graph. In the case of unweighted graphs, each element $\mathbf{A}_{ij}$ is equal to 1 if there is an edge



(a) Undirected Graph (b) Directed Graph (c) Weighted Graph

Figure 1.1: Illustration of different graph types. (a) A simple graph is an unweighted and undirected; (b) For a directed graph, edges indicate an orientation, and (c) edges can have weights in a weighted graph.

connecting node i and node j, and it is 0 otherwise. For weighted graphs, this matrix contains real values to represent edge weights. In undirected graphs, the adjacency matrix is symmetric.

Degree matrix The degree matrix $\mathbf{D} = \mathrm{diag}(d_i)$ is a diagonal matrix in which each diagonal entry d_i corresponds to the degree of node i. The degree of a node represents the number of edges incident to that node, reflecting its connectivity within the graph.

Laplacian matrix For a graph $\mathbb{G}$, the Laplacian matrix $\mathbf{L} \in \mathbb{R}^{N \times N}$ is defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \tag{1.1}$$

where $\mathbf{D}$ represents the degree matrix, and $\mathbf{A}$ is the adjacency matrix.

These matrices provide a foundation for various graph-based algorithms and techniques, enabling the exploration of intricate relationships within graph-structured data.

1.5.2 Graph Embedding

Graph embedding is a fundamental technique in the field of graph-based data analysis. It involves transforming complex graph structures into continuous, lower-dimensional vector representations while preserving essential structural and relational information. These embeddings enable the application of various machine learning and deep learning techniques to graph data, facilitating tasks such as node classification, link prediction, and anomaly detection. Effective graph embeddings play a pivotal role in enhancing the interpretability and computational efficiency of graph-based algorithms, making them a cornerstone of contemporary graph representation learning methodologies.

Node Embedding. The goal of graph representation learning is to map each node in the graph to a vector in a low-dimensional vector space while preserving the structure of the original graph. Given a graph $\mathbb{G}$, the aim of graph representation learning is to learn a mapping $f : \mathcal{V} \rightarrow \mathbb{R}^P$, where $P \ll N$ is the dimension of the embedding. Node embedding can benefit node related tasks

such as node classification, and node clustering. An illustration of node embedding is shown in Figure 1.2.

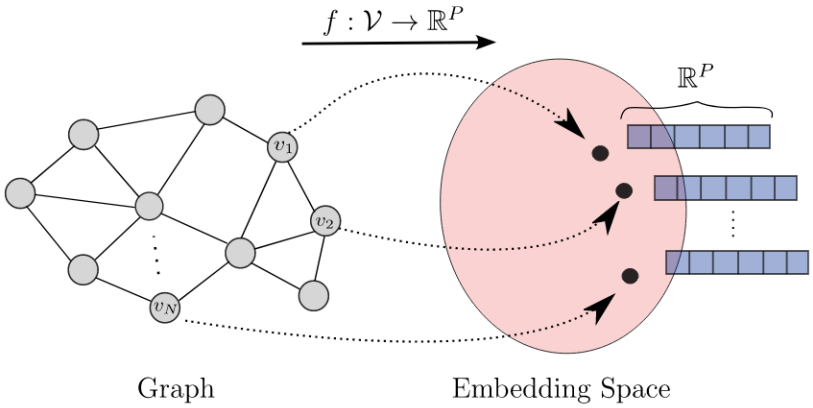


Figure 1.2: Schematic diagram of node embedding.

1.5.3 Graph Embedding Applications

The embedding vectors generated through graph embedding techniques have proven to be invaluable across various domains, serving as the foundation for numerous applications. This section explores several key applications, including node classification, link prediction, and anomaly detection.

Node Classification. Node classification stands as one of the most prominent research directions within the realm of graph analysis, primarily due to its widespread application scenarios. The fundamental goal of node classification is to predict specific class labels for unlabeled nodes in a given graph by leveraging the information encoded within the graph structure [53]. This task finds diverse applications, such as in academic citation networks [54]. In these networks, node classification serves to predict the research topics to which each article belongs. Node classification techniques enable automated categorization of nodes within a graph, thereby facilitating decision-making processes, recommendation systems, and knowledge discovery.

Link prediction. The task of link prediction within graph focuses on determining the likelihood of a connection between two nodes in the graph. This problem is often tackled by analyzing the local and global structural properties of the graph surrounding the two target nodes. The key idea is to investigate the specific features of the graph structure that promote the establishment of links between the two central nodes. By leveraging this understanding, link prediction models aim to infer missing or potential connections within the graph, offering valuable insights for network expansion, recommendation systems, and relational data analysis [55, 56].

Anomaly detection. Anomaly detection serves the purpose of identifying unusual nodes, edges, or entire graphs that deviate from typical behavior. In transaction networks, for example, anomalous nodes may exhibit sudden, large transactions or extensive connections compared to their counterparts [57]. Detecting such anomalies is crucial for fraud detection, network security miao2020attack, and outlier identification across various domains. Anomaly detection techniques leverage graph embeddings to uncover irregularities within graph-structured data. By extracting expressive representations that facilitate the clear separation of graph anomalies from normal objects or the learning of deviating patterns among anomalies, these techniques enhance data integrity and security measures [58].

1.5.4 Graph Neural Networks

Graph Neural Networks (GNNs) have emerged as powerful models for processing graph-structured data, finding applications in various domains such as social networks and recommendation systems. GNNs excel at generating informative node embeddings by aggregating information from neighboring nodes [3]. These networks can be employed in supervised, semi-supervised, and unsupervised learning frameworks.

- **Supervised Learning.** In a supervised learning framework, GNN models are trained on labeled graph datasets, where nodes have associated labels or target values. The primary objective is to learn a mapping that can accurately predict the correct labels or values for nodes or edges in the graph. GNNs leverage the labeled nodes to propagate information through the graph, enabling them to learn how to make predictions based on both the graph structure and the provided labels. This paradigm finds applications in tasks such as node classification, where GNNs are employed to categorize nodes based on their structural context and labels.

- **Semi-Supervised Learning.** In a semi-supervised learning framework, the dataset consists of both labeled and unlabeled nodes within the graph. Semi-supervised GNNs offer a versatile approach as they leverage the labeled data for supervised learning tasks while also utilizing the unlabeled data to propagate information and improve their representations. This proves to be particularly valuable when obtaining labeled data is costly or limited, as semi-supervised GNNs can effectively make use of unlabeled data to enhance their performance [59]. They find applications in tasks such as node classification, where they can predict labels for unlabeled nodes based on the collective knowledge learned from both labeled and unlabeled nodes.

- **Unsupervised Learning.** In an unsupervised learning framework, the dataset typically contains only the graph's structural information, and explicit node labels or values are absent. The objective in unsupervised learning is to learn meaningful representations of nodes or edges based solely on the graph's topology and connectivity patterns. Unsupervised GNNs employ techniques like graph autoencoders or variational graph encoders to learn unsupervised embeddings that capture latent structures and relationships within the graph [60]. These learned embeddings can be subsequently utilized for various downstream tasks or for exploring the inherent graph properties.

This section explores several key components that make up GNNs, including graph convolution layers, pooling layers, skip connections, and activation functions . Each of these components plays a crucial role in enabling GNNs to effectively process and learn from graph data, making them indispensable tools for various machine learning tasks.

Graph Convolution Layers. At the core of GNNs are graph convolution layers, which perform a form of convolution operation on the graph. These layers are responsible for propagating information across the graph, enabling GNNs to learn meaningful representations of nodes by aggregating information from their neighboring nodes. GNNs can have multiple such layers, allowing for increasingly abstract representations of nodes. In essence, these layers work by iteratively updating the node features. At each layer, information from neighboring nodes is gathered and combined, allowing the model to build a richer understanding of each node's context within the graph.

Pooling Layers. Graph pooling is a technique used in GNNs to reduce the size of a graph while preserving its important structural and semantic information. The primary goal of graph pooling is to downsample the graph while retaining its essential features, which can be crucial for tasks such as graph classification and node-level classification. There are several categories of graph pooling techniques, each with its own approach and advantages. One of the common categories of graph pooling is hierarchical graph pooling operators. Hierarchical graph pooling operators group proximal nodes together using graph clustering methods, effectively coarsening the original graph into a new graph with a coarser granularity. There are three main approaches to hierarchical graph pooling [3]. The first approach involves leveraging graph clustering algorithms to partition the graph into disjoint clusters, which are then consolidated into super-nodes with interconnections becoming super-edges [61]. Node representations in super-nodes are aggregated using functions such as max or average pooling. The second technique focuses on learning a soft cluster assignment matrix, achieved by models that utilize node features and the adjacency matrix through probabilistic inference methods [62]. In the third approach, top-ranked nodes are selected to construct the

coarser-grained graph, often utilizing self-attention scores for node ranking [42, 44].

Activation Function. The purpose of the activation function is to introduce non-linearity into the output of a neuron in a deep neural network. It decides whether a neuron should be activated or not, and hence only the activated features are carried forward into the next layer.

- **ReLU Function:** A commonly used activation function is the Rectified Linear Unit (ReLU) defined as

$$\text{ReLU}(x) = \max(0, x), \tag{1.2}$$

  where x is the input to a neuron. ReLU is highly effective in addressing the vanishing gradient problem, a challenge often encountered during training in deep neural networks. This issue arises when gradients become exceedingly small as they are propagated backward through numerous layers, making it difficult for the network to learn effectively. ReLU combats this problem by mapping negative values to zero, thus preventing the rapid vanishing of gradients associated with other activation functions, such as sigmoid or tanh. By maintaining positive values, ReLU contributes to faster and more efficient training, promoting the convergence of deep neural networks.

- **Softmax Function:** The softmax activation function is a popular activation function used primarily in the output layer of neural networks for multi-class classification problems. It is designed to convert a vector of raw scores or logits into a probability distribution over multiple classes. The softmax function takes as input a vector of real numbers and transforms them into a probability distribution where each element represents the likelihood of belonging to a particular class. Suppose the predicted output from the model for all classes is the C-dimensional vector $\mathbf{z} = (z_1, \ldots, z_C)^\intercal$, where C is the total number of classes. The softmax function maps $\mathbf{z}$ into another C-dimensional vector $\hat{\mathbf{y}} = (\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_C)^\intercal$ of probabilities summing up to one:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}) = \left(\frac{e^{z_1}}{\sum_{c=1}^{C} e^{z_c}}, \ldots, \frac{e^{z_C}}{\sum_{c=1}^{C} e^{z_c}} \right), \tag{1.3}$$

  These predicted probabilities provide useful information about the model's confidence in prediction. The higher the probability for the predicted class, the more confident the prediction is.

Skip Connections. A skip connection in deep architectures means skipping some layers in the neural network and feeding one layer's output as an input to the next layers, not just the immediate

next layer. Skip connections alleviate the vanishing gradient, which hampers gradient flow in deep networks, and over-smoothing which can lead to a loss of information in node representations. Skip connections offer an effective remedy, substantially enhancing the accuracy and stability of the training process.

In the context of a multi-layer GNN with L layers, skip connections can be thought of as bridges between different layers, helping to retain and transfer information effectively. These connections can be applied after specific graph convolutional layers, combining the current embeddings with those from previous layers. Specifically, $\mathbf{H}^{(\ell)}$ is the input feature matrix of the ℓ-th layer for $\ell = 0, \ldots, L-1$ and the input of the first layer is the initial feature matrix $\mathbf{H}^{(0)} = \mathbf{X}$. There are four types of skip connections [63]:

- **Residual Connection:** This connection combins current layer embeddings with those from the previous layer [64]

$$\mathbf{H}^{(\ell+1)} = (1-\alpha)\mathbf{H}^{(\ell+1)} + \alpha\mathbf{H}^{(\ell)}, \tag{1.4}$$

  where α is a hyperparameter to weigh the contributions of node features from the current layer and previous layer.

- **Initial Connection:** This connection combines current layer embeddings with the initial node features, represented by $\mathbf{X}$ [32, 64, 65]

$$\mathbf{H}^{(\ell+1)} = (1-\alpha)\mathbf{H}^{(\ell+1)} + \alpha\mathbf{X}. \tag{1.5}$$

- **Dense Connection:** It combines embeddings from all previous layers during forward propagation [64]

$$\mathbf{H}^{(\ell+1)} = \text{COM}(\{\mathbf{H}^{(k)}, 0 \leq k \leq \ell+1\}). \tag{1.6}$$

  The combination functions (COM) offer diverse methods to aggregate information across layers [63]. These functions include concatenation, which combines node representations at each layer through simple concatenation, max pooling, which selects the maximum value along each dimension of feature vectors from previous layers, and an attention mechanism that calculates attention scores to weight contributions from previous layers before summation.

- **Jumping Connection:** This connection is a simplified case of dense connection and only applied at the end of whole forward propagation process to combine the node features from all previous layers [66, 67]

$$\mathbf{H}^{(L)} = \text{COM}(\{\mathbf{H}^{(k)}, 0 \leq k \leq L\}). \tag{1.7}$$

An illustration of residual connection and initial connection is shown in Figure 1.3.

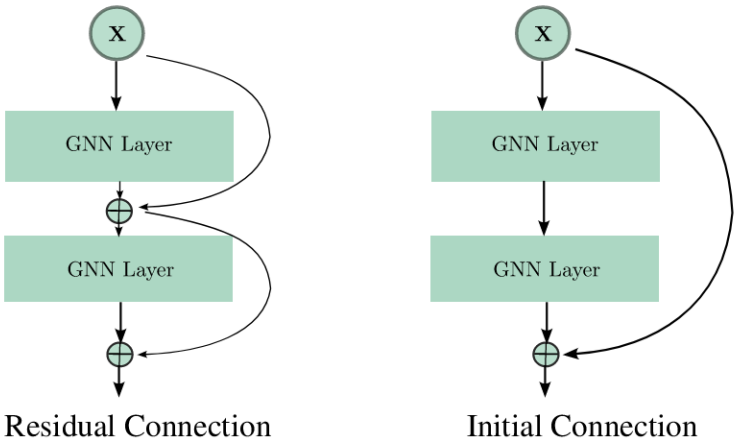


Figure 1.3: Illustration of residual connection and initial connection.

1.5.5 Graph Convolutional Networks

Existing GNNs fall into two main categories: spatial methods and spectral methods. The former define graph convolution in the node domain as a weighted average function over neighboring nodes similar to the idea of convolution in traditional convolutional neural networks. The latter define graph convolution in the graph Fourier domain using the eigenvectors of the graph Laplacian matrix.

Spatial Methods. Convolution in spatial-based graph neural networks is defined directly in the node domain based on a node's spatial relations (i.e. neighbors). The main idea is that a node representation is updated by aggregating information from its neighboring nodes. The spatial graph convolutional operation essentially propagates node information along edges. The information that passes between neighbors and the central node in the graph is referred to as messages. During each message-passing iteration in a graph neural network, the embedding $\mathbf{h}_i^{(\ell)}$ of node $i \in \mathcal{V}$ is updated according to information aggregated from its graph neighborhood $\mathcal{N}_i$ using two neural networks AGGREGATE and UPDATE as follows:

$$\mathbf{m}_{\mathcal{N}_i}^{(\ell)} = \text{AGGREGATE}^{(\ell)}(\{\mathbf{h}_j^{(\ell)} : \ j \in \mathcal{N}_i\}). \tag{1.8}$$

and

$$\mathbf{h}_i^{(\ell+1)} = \text{UPDATE}^{(\ell)}(\mathbf{h}_i^{(\ell)}, \mathbf{m}_{\mathcal{N}_i}^{(\ell)}), \tag{1.9}$$

where $\mathbf{m}_{\mathcal{N}_i}^{(\ell)}$ is the message aggregated from the neighborhood of node i, and $\mathbf{h}_i^{(\ell+1)}$ is the embedding of node i at the $(\ell+1)$-layer.

Spectral Methods. Spectral techniques define graph convolution using graph signal processing.

- **Graph Fourier Transform:** We can generalize a convolutional network for a spectral network via graph Fourier transform based on the graph Laplacian matrix [1]. Suppose an input vector $\mathbf{x} \in \mathbb{R}^N$ is a signal defined on a graph $\mathbb{G}$ with N nodes. If $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix associated with a graph $\mathbb{G}$ and $\mathbf{D}$ is the diagonal degree matrix, then the normalized graph Laplacian matrix is defined as $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The normalized Laplacian $\mathbf{L}$ admits an eigendecomposition given by $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\intercal$, where $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_N)$ is an orthonormal matrix whose columns constitute an orthonormal basis of eigenvectors and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_N)$ is a diagonal matrix comprised of the corresponding eigenvalues such that $0 = \lambda_1 \leq \ldots \leq \lambda_N \leq 2$ in ascending order [68]. The graph Fourier transform of a signal $\mathbf{x} \in \mathbb{R}^N$ is defined as $\mathcal{F}(\mathbf{x}) = \hat{\mathbf{x}} = \mathbf{U}^\intercal\mathbf{x} \in \mathbb{R}^N$, and its inverse is given by $\mathcal{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}$.

- **Spectral Filtering of Graph Signals:** The convolution of a graph filter $\mathbf{g}$ and a graph signal $\mathbf{x}$ is defined as

$$\mathbf{g} * \mathbf{x} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{g}) \odot \mathcal{F}(\mathbf{x})) = \mathbf{U}(\mathbf{U}^\intercal\mathbf{g} \odot \mathbf{U}^\intercal\mathbf{x}), \tag{1.10}$$

  where $\odot$ denotes element-wise multiplication. Hence, applying a spectral graph filter $\mathbf{g}_{\boldsymbol{\theta}}$ on a graph signal $\mathbf{x}$ yields

$$\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{L})\mathbf{x} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\intercal)\mathbf{x} = \mathbf{U}\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{\Lambda})\mathbf{U}^\intercal\mathbf{x}, \tag{1.11}$$

  where $\boldsymbol{\theta}$ is a vector of learnable parameters. However, there are three limitations that prohibit the spectral filter from being used in practice: the filter is not localized, the learning complexity is $\mathcal{O}(N^2)$ due to matrix-vector multiplication, and the number of parameters depends on the input size. To tackle these limitations, the spectral filter can be approximated using Chebyshev polynomials as follows

$$\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k T_k(\hat{\mathbf{\Lambda}}), \tag{1.12}$$

  where the Chebyshev polynomials are defined recursively by

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \quad \text{with } T_0 = 1 \text{ and } T_1 = x \tag{1.13}$$

  and $\hat{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{\text{max}} - \mathbf{I}$ is a diagonal matrix of scaled eigenvalues with λ_{max} denoting the largest eigenvalue of the Laplacian matrix. Hence, the cost of the resulting filtering operation is reduced to $\mathcal{O}(K|\mathcal{E}|)$.

Figure 1.4 illustrates a graph neural network featuring multiple graph convolutional layers. Each graph convolutional layer captures the hidden representation of individual nodes by aggregating feature information from its neighbors. After feature aggregation, a non-linear transformation is applied to the resulted outputs. By stacking multiple layers, each node's final hidden representation incorporates information from an extended neighborhood.

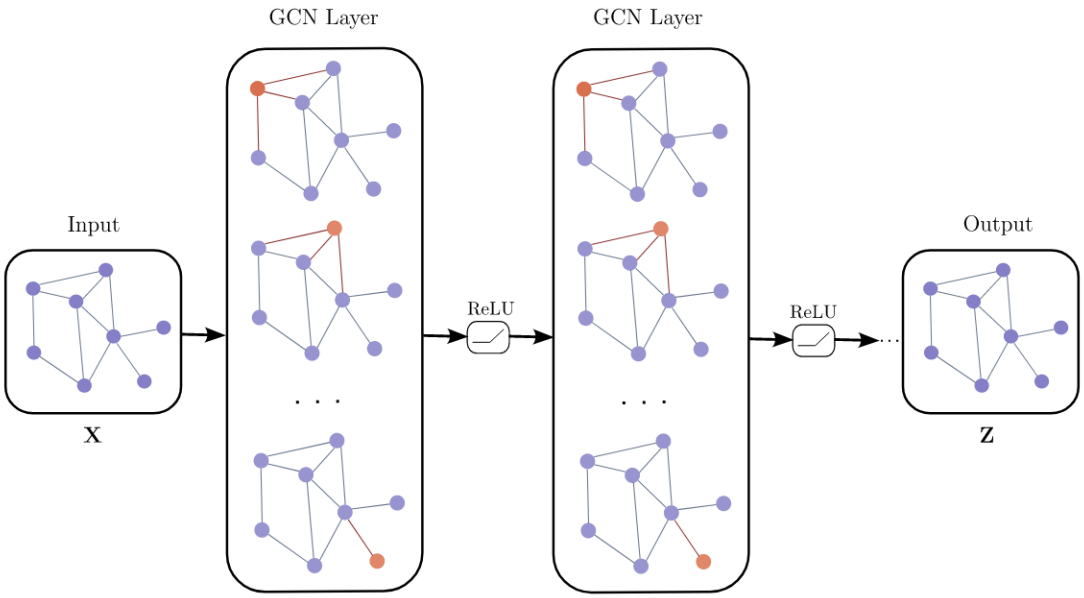


Figure 1.4: Schematic diagram of graph convolutional neural network with multiple graph convolutional layers. In this diagram, the input $\mathbf{X}$ represents the initial feature matrix of node attributes, and the output $\mathbf{Z}$ is the latent graph representation from the final network layer.

1.6 Overview and Contributions

The organization of this thesis is as follows:

- In Chapter 1, we begin with the basic concepts which we refer to throughout the thesis. Then, we present the motivations and the goals of this research, followed by the problem statement, the objectives of this study, and literature review. We also present an overview of graph theory basics, graph embedding, graph embedding applications, graph neural networks, and graph convolutional networks.

- In Chapter 2, we introduce an anisotropic graph convolutional network for semi-supervised node classification [4]. We integrate a nonlinearity term into the graph convolution to make it anisotropic, resulting in a feature-preserving graph neural network that captures informative node features while mitigating over-smoothing. Our approach draws inspiration from

the success of anisotropic diffusion in image and geometry processing, learning nonlinear representations based on local graph structure and node features.

- In Chapter 3, we propose a graph fairing convolutional network for semi-supervised anomaly detection [5]. It consists of a graph convolution module for aggregating information from immediate node neighbors and a skip connection between the initial feature matrix and each hidden layer, allowing the model to retain important information from the original data throughout the network's layers. The update rule of the proposed model is derived from the iterative solution of the implicit fairing equation using the Jacobi method. This helps in smoothing out the graph and improving the robustness of the model.

- In chapter 4, we design a graph encoder-decoder architecture for unsupervised anomaly detection on graph-structure data [6]. We introduce a locality-constrained pooling strategy that preserves local graph structures during the pooling process without relying on learnable parameters. This parameter-free approach enhances flexibility and adaptability. The proposed model employs a multi-layer graph convolutional network encoder, followed by the pooling layers. In the decoding phase, an unpooling operation is performed, followed by a graph deconvolutional network decoder. Our model also incorporates denoising operations using spectral graph wavelets. This step helps reduce the impact of noise during the decoding stage, enhancing the overall robustness and accuracy of the model.

- In Chapter 5, we present a summary of the contributions of this thesis and limitations, and we also outline several directions for future research in this area of study.

CHAPTER 2

Anisotropic Graph Convolutional Network for Semi-supervised Learning

Graph convolutional networks learn effective node embeddings that have proven to be useful in achieving high-accuracy prediction results in semi-supervised learning tasks, such as node classification. However, these networks suffer from the issue of over-smoothing and shrinking effect of the graph due in large part to the fact that they diffuse features across the edges of the graph using a linear Laplacian flow. This limitation is especially problematic for the task of node classification, where the goal is to predict the label associated with a graph node. To address this issue, we propose an anisotropic graph convolutional network for semi-supervised node classification by introducing a nonlinear function that captures informative features from nodes, while preventing over-smoothing. The proposed framework is largely motivated by the good performance of anisotropic diffusion in image and geometry processing, and learns nonlinear representations based on local graph structure and node features. The effectiveness of our approach is demonstrated on three citation networks and two image datasets, achieving better or comparable classification accuracy results compared to the standard baseline methods.

2.1 Introduction

Graphs are ubiquitous in a wide array of application domains, ranging from social networks [69–71] and transportation systems [72] and cyber-security [73] to brain networks [74] graph signal processing [68], and video analysis [75–77]. They provide a flexible way to inherently represent

real-world entities as a set of nodes and their interactions as a set of links/edges. This interconnection of entities and their pairwise relationships forms a graph structure when visualized.

With the prevalence and increasing proliferation of graph-structured data in real-world applications, there has been a surge of interest in developing efficient representations of graphs. Network embedding has recently emerged as a powerful paradigm for representing and analyzing graph-structured data [7–10]. The idea is to learn low-dimensional embedding vectors, such that both structural and semantic information are captured. These learned embeddings can then be used as input to various machine learning algorithms for downstream tasks, such as link prediction, visualization, recommendation, community detection, and node classification. The latter task is the focus of this chapter. The objective of node classification is to predict the most probable labels of nodes in a graph [13]. In a social network, for instance, we want to predict user labels such as their interest, beliefs or other characteristics [69], while in a citation network, we want to classify documents based on their topics.

There is a sizable body of literature on network embedding that has centered around the use of random walks and neural language models to learn effective low-dimensional embedding vectors of graph nodes [11–13]. Perozzi *et al.* [11] introduce DeepWalk, a deep learning based framework that learns latent representations of nodes in a graph by leveraging local information obtained from truncated random walks. Each random walk is treated as a sentence that is fed into the skip-gram language model [14], which maximizes the co-occurrence probability among the words that appear within a window in a sentence. Another popular approach that also uses random walks is node2vec [12], a semi-supervised algorithm for feature learning in graphs that can be regarded as a generalization of DeepWalk. While DeepWalk performs a uniform random walk, node2vec uses a second-order random walk approach to generate network neighborhoods for nodes via breadth-first and depth-first sampling strategies. However, node2vec involves a number of parameters that require fine-tuning for each dataset and each task.

In recent years, the advent of deep learning has sparked groundswell of interest in the adoption of graph neural networks (GNNs) for learning latent representations of graphs [1, 15–17, 78–83]. A plethora of GNNs is based on convolutional neural networks (CNNs) and network embedding. Defferrard *et al.* [17] introduce the Chebyshev network (ChebyNet), an efficient spectral-domain graph convolutional neural network that uses recursive Chebyshev polynomial spectral filters to avoid explicit computation of the Laplacian eigenvectors. These filters are localized in space, and the learned weights can be shared across different locations in a graph. An efficient variant of GNNs is graph convolutional networks (GCNs) [1], which is an upsurging semi-supervised graph-based deep learning framework that uses an efficient layer-wise propagation rule based on a first-

order approximation of spectral graph convolutions. Hamilton *et al.* [81] propose GraphSAGE, a general inductive framework that generates embeddings by sampling and aggregating features from the local neighborhood of a graph node. Veličković *et al.* [82] present the graph attention network, which is a graph-based neural network architecture that uses an attention mechanism to assign self-attention scores to neighboring node embeddings. These scores indicate the importance of graph nodes to their corresponding neighbors on the feature aggregation process. Xu *et al.* [83] present theoretical foundations for analyzing the expressive power of GNNs in an effort to capture different graph structures, and develop a graph isomorphism network whose goal is to map isomorphic graphs to the same representation and non-isomorphic ones to different representations.

While graph convolutional networks have achieved state-of-the-art performance on semi-supervised node classification tasks, they tend, however, to oversmooth the learned feature embeddings of graph nodes [84]. This is due largely to the fact that the graph convolution of the GCN model is a special form of graph Laplacian smoothing, which repeatedly and simultaneously adjusts the location of each graph node to the weighted average (i.e. geometric center) of its neighboring nodes. This averaging process causes an oversmoothing effect on the graph, as it reduces the high-frequency graph information and tends to flatten the graph. Moreover, oversmoothing causes features at nodes within each connected component to converge to the same value. Hence, nodes from different classes may be predicted to have similar labels, resulting in misclassification errors. Another drawback of Laplacian smoothing is shrinkage of the graph, as repeated iterations of the smoothing process causes the shrinking effect.

In this chapter, we propose an anisotropic graph convolutional network (AGCN), which adopts the concept of anisotropic diffusion, previously used in image and geometry processing tasks, to overcome the aforementioned issues. The idea behind our proposed model is to integrate a nonlinearity term into the graph convolution to make it non-linear and/or anisotropic, resulting in a feature-preserving graph neural network. The main contributions of this work can be summarized as follows:

- We introduce a novel anisotropic graph convolutional network for semi-supervised learning.
- We learn efficient representations for node classification in an end-to-end fashion.
- We demonstrate that AGCN can be integrated into existing graph-based convolutional networks for semi-supervised learning using both co-training and self-training.
- Our extensive experimental results show competitive or superior performance of AGCN over standard baseline methods on several benchmark datasets.

The rest of this chapter is organized as follows. In Section 2, we review important relevant work. In Section 3, we present the problem formulation and propose an anisotropic graph convolutional network architecture for semi-supervised learning. We discuss in detail the main components of the proposed framework and analyze the model complexity. In Section 4, we present experimental results to demonstrate the competitive performance of our approach on five standard benchmark datasets, including three citations networks and two image datasets. Finally, we conclude in Section 5 and point out future work directions.

2.2 Related Work

The basic goal of node classification is to predict the most probable labels of nodes in a graph. Graph convolutional networks (GCNs) have recently become the de facto model for semi-supervised node classification [1]. GCN uses an efficient layer-wise propagation rule, which is based on a first-order approximation of spectral graph convolutions. The feature vector of each graph node is updated by essentially applying a weighted sum of the features of its neighboring nodes. Monti *et al.* [18] present a mixture of networks (MoNet) model, a spatial-domain graph convolutional neural network that employs a mixture of Gaussian kernels with learnable parameters to model the weight function of pseudo-coordinates, which are associated to the neighboring nodes of each graph node. Liao *et al.* [19] propose the Lanczos network (LanczosNet), which employs the Lanczos algorithm to construct low-rank approximations of the graph Laplacian in order to facilitate efficient computations of matrix powers. Veličković *et al.* [85] present deep graph infomax, an unsupervised graph representation learning approach, which relies on training an encoder model to maximize the mutual information between local and global representations in graphs. Xu *et al.* [20] introduce a graph wavelet neural network, which is a GCN-based architecture that uses spectral graph wavelets in lieu of graph Fourier bases to define a graph convolution. Despite the fact that spectral graph wavelets can yield localization of graph signals in both spatial and spectral domains, they require explicit computation of the Laplacian eigenbasis, leading to a high computational complexity, especially for large graphs. In order to avoid this issue, recursive Chebyshev polynomial spectral filters can be employed.

While GCNs have shown great promise, achieving state-of-the-art performance on semi-supervised node classification, they are prone to oversmoothing the node features. In fact, the neighborhood aggregation scheme (i.e. graph convolution) of GCN is tantamount to applying Laplacian smoothing [84], which replaces each graph node with the average of its immediate neighbors. Therefore, repeated application of GCN yields smoother and smoother versions of the initial node features as the number of the network's layers increases. As a result, the node fea-

tures in deeper layers will eventually converge to the same value, and hence become too similar across different classes. Wu *et al.* [21] introduce a simple graph convolution by removing the non-linear transition functions between the layers of graph convolutional networks and collapsing the resulting function into a single linear transformation via the powers of the normalized adjacency matrix with added self-loops for all graph nodes. However, this simple graph convolution acts as a low-pass filter, which attenuates all but the zero frequency, causing oversmoothing. Recently, significant strides have been made toward remedying the issue of oversmoothing in GCNs [22,23]. Xu *et al.* [22] propose jumping knowledge networks, which employ dense skip connections to connect each layer of the network with the last layer to preserve the locality of node representations in order circumvent oversmoothing. More recently, a normalization layer, which helps avoid oversmoothing by preventing learned representations of distant nodes from becoming indistinguishable, has been proposed in [23]. This normalization layer is performed on intermediate layers during training, and the aim is to apply smoothing over nodes within the same cluster while avoiding smoothing over nodes from different clusters. While these approaches have shown slightly improved results using deeper GCNs, the issue of oversmoothing still remains a daunting task, as performance gains do not usually reflect the benefits of increasing the network depth.

2.3 Method

In this section, we describe the problem statement and introduce an anisotropic graph convolutional network for semi-supervised node classification. In particular, we examine the main building blocks of the proposed network architecture and analyze the complexity of the model. We also show that our proposed aggregation scheme seamlessly incorporates both the graph structure and the node features without sacrificing performance in an effort to alleviate oversmoothing of the learned node representations.

2.3.1 Problem Formulation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V} = \{1, \dots, N\}$ is the set of N nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We denote by $\mathbf{A} = (\mathbf{A}_{ij})$ an $N \times N$ adjacency matrix (binary or real-valued) whose (i, j)-th entry $\mathbf{A}_{ij}$ is equal to the weight of the edge between neighboring nodes i and j, and 0 otherwise. We also denote by $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^\intercal$ an $N \times F$ feature matrix of node attributes, where $\mathbf{x}_i$ is an F-dimensional row vector for node i.

Learning latent representations of nodes in a graph aims at encoding the graph structure into low-dimensional embeddings, such that both structural and semantic information are captured.

More precisely, the purpose of network/graph embedding is to learn a mapping $\varphi : \mathcal{V} \to \mathbb{R}^P$ that maps each node i to a P-dimensional vector $\mathbf{z}_i$, where $P \ll N$. These learned node embeddings can then be used as input to learning algorithms for downstream tasks, such as node classification.

Given the labels of a subset of the graph nodes (or their corresponding final output embeddings), the objective of semi-supervised learning is to predict the unknown labels of the other nodes. More specifically, let $\mathcal{D}_K = \{(\mathbf{z}_i, y_i)\}_{i=1}^{K}$ be the set of labeled final output node embeddings $\mathbf{z}_i \in \mathbb{R}^P$ with associated known labels $y_i \in \mathcal{Y}_K$, and $\mathcal{D}_U = \{\mathbf{z}_i\}_{i=K+1}^{K+U}$ be the set of unlabeled final output node embeddings, where $K + U = N$. Then, the problem of semi-supervised node classification is to learn a classifier $f : \mathcal{V} \to \mathcal{Y}_K$. That is, the goal is to predict the labels of the set $\mathcal{D}_U$.

It is important to note that for multi-class classification problems, the label of each node i (or its final output embedding $\mathbf{z}_i$) in the labeled set $\mathcal{D}_K$ can be represented as a C-dimensional one-hot vector $\mathbf{y}_i \in \{0, 1\}^C$, where C is the number of classes.

2.3.2 Proposed Approach

Graph convolutional networks learn a new feature representation for each node such that nodes with the same labels have similar features [1]. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ and feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, the layer-wise feature diffusion rule of an L-layer GCN is given by

$$\mathbf{S}^{(\ell)} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(\ell)}, \quad \ell = 0, \ldots, L-1, \tag{2.1}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-added loops, $\mathbf{I}_N$ is the identity matrix, $\tilde{\mathbf{D}} = \mathrm{diag}(\tilde{d}_i)$ is the diagonal degree matrix whose i-th diagonal entry is the degree of node i with added self-loops, and $\mathbf{H}^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$ is the input feature matrix of the ℓ-th layer with F_ℓ feature maps. The input of the first layer is the original feature matrix $\mathbf{H}^{(0)} = \mathbf{X}$.

Using the feature diffusion rule of GCN is tantamount to applying a weighted sum of the features of neighboring nodes normalized by their degrees, which essentially performs Laplacian smoothing on the graph [21, 84]. In other words, the smooth feature matrix $\mathbf{S}^{(\ell)}$ is obtained by applying Laplacian smoothing to the input feature matrix at the ℓ-th layer. Intuitively, the Laplacian flow repeatedly and simultaneously adjusts the location of each graph node to the geometric center of its neighboring nodes. Although the Laplacian smoothing flow is simple and fast, it produces, however, the shrinking effect and an oversmoothing result.

Motivated by the good performance of anisotropic diffusion in image and mesh denoising [86–88], and in an effort to tackle the issues of oversmoothing and shrinking effect of GCN, we propose an anisotropic graph convolutional network (AGCN) for semi-supervised node classification by incorporating a nonlinear smoothness term into the GCN feature diffusion rule. This nonlinearity

term, which quantifies the dissimilarity between learned node embeddings, plays a pivotal role in preventing these learned representations from becoming increasingly similar, and hence alleviates the issue of oversmoothing. In addition, it tackles the shrinking effect by precluding the learned node representations from converging to the same value.

Anisotropic feature diffusion. We define a layer-wise anisotropic feature diffusion rule for node features in the ℓ-th layer as follows:

$$\mathbf{G}^{(\ell)} = \Big(1 - \exp\big(-\beta \operatorname{tr}^2(\mathbf{H}^{(\ell)\intercal}\tilde{\mathbf{L}}\mathbf{H}^{(\ell)})\big)\Big)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(\ell)}, \tag{2.2}$$

where β is an nonnegative hyper-parameter that is often fine-tuned via grid search, and $\operatorname{tr}(\mathbf{H}^{(\ell)\intercal}\tilde{\mathbf{L}}\mathbf{H}^{(\ell)})$ is a Laplacian smoothness term given by

$$\operatorname{tr}(\mathbf{H}^{(\ell)\intercal}\tilde{\mathbf{L}}\mathbf{H}^{(\ell)}) = \frac{1}{2}\sum_{i,j=1}^{N}\tilde{\mathbf{A}}_{ij}\|\mathbf{h}_i^{(\ell)} - \mathbf{h}_j^{(\ell)}\|^2, \tag{2.3}$$

with $\mathbf{H}^{(\ell)} = (\mathbf{h}_1^{(\ell)}, \ldots, \mathbf{h}_N^{(\ell)})^\intercal$; $\mathbf{h}_i^{(\ell)}$ is an F_ℓ-dimensional hidden representation (embedding) vector of the i-th node at the ℓ-th layer, $\operatorname{tr}(\cdot)$ denotes the trace operator, $\|\cdot\|$ denotes the 2-norm, and $\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}}$ is an $N \times N$ Laplacian matrix.

For each pair of similar embeddings at the ℓ-th layer, the Laplacian smoothness term enforces their predictions to be close to each other. The strength of this smoothness is determined by the weight of the edge between neighboring nodes, meaning that connected nodes will have similar predictions.

The Laplacian smoothness term plays a crucial role not only in explicitly taking into consideration the correlation between embeddings, but also in preserving the locality of nodes to be embedded. In other words, two nodes or their attributes $\mathbf{x}_i$ and $\mathbf{x}_j$ that are close to each other in the original graph (i.e. adjacent nodes in $\mathcal{V}$) are encoded as embeddings $\mathbf{h}_i^{(\ell)}$ and $\mathbf{h}_j^{(\ell)}$ that are more likely to be close to each other in the embedding vector space. Such a locality-preserving property is of paramount importance in classification tasks.

The nonlinearity term $1 - \exp\big(-\beta \operatorname{tr}^2(\mathbf{H}^{(\ell)\intercal}\tilde{\mathbf{L}}\mathbf{H}^{(\ell)})\big)$ can be regarded as an oversmooting "stopping" function. In fact, it only incurs a small penalty when similar nodes with a large smoothness strength $\tilde{\mathbf{A}}_{ij}$ have different learned embeddings. Hence, it reduces the oversmoothing effect on the learned graph features.

Anisotropic aggregation procedure. The anisotropic feature diffusion rule can be written in vector form as follows:

$$\mathbf{g}_i^{(\ell)} = \sum_{j=1}^{N}\alpha_{ij}^{(\ell)}\mathbf{h}_j^{(\ell)} \tag{2.4}$$

where $\mathbf{g}_i^{(\ell)}$ is the i-th row of $\mathbf{G}_i^{(\ell)}$, and $\alpha_{ij}^{(\ell)}$ is the layer-wise weight coefficient given by

$$\alpha_{ij}^{(\ell)} = \Big(1 - \exp\big(-\beta \operatorname{tr}^2(\mathbf{H}^{(\ell)\intercal}\tilde{\mathbf{L}}\mathbf{H}^{(\ell)})\big)\Big)\frac{\tilde{\mathbf{A}}_{ij}}{\sqrt{\tilde{d}_i\tilde{d}_j}} \tag{2.5}$$

which is equal to the nonlinearity term times the weight of the GCN neighborhood aggregation. The anisotropic aggregation scheme is illustrated in Figure 2.1. Notice how features are propagated from 1-hop (i.e. immediate) neighbors to multi-hop (i.e. distant) neighbors as the number of layers increases.

Unlike the GCN weight which only takes into account the topological structure of the graph, our proposed anisotropic feature diffusion rule seamlessly leverages both the topological structure and nodal attributes for aggregating node representations. Also, it is important to note that compared to existing neighborhood aggregation approaches [1, 81, 82], the weight coefficient $\alpha_{ij}^{(\ell)}$ of our aggregation scheme is layer-aware, and hence prevents the learned representations from becoming indistinguishable thanks to the synergy between the nonlinearity term that helps mitigate oversmoothing of the node features and the GCN weight that captures the graph structure.

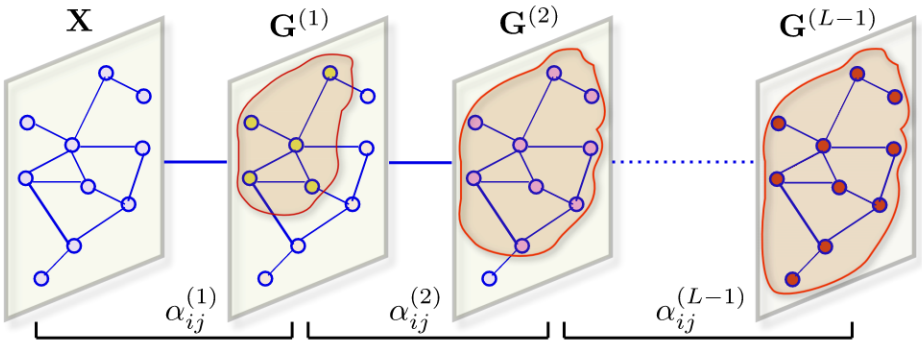


Figure 2.1: Schematic layout of the anisotropic aggregation procedure.

As illustrated in Figure 2.1, the proposed anisotropic feature diffusion rule follows a neighborhood aggregation or a message passing algorithm to learn a node representation by propagating representations of its immediate neighbors, where the latent representation of each node is initialized to the node's input features. The latent representation of each graph node at a given layer is defined as a weighted sum of its immediate neighbors' representations from the previous layer. As the number of layers increases, the node features are propagated to higher-order neighborhoods.

Learning embeddings. Given the anisotropically smooth feature matrix $\mathbf{G}^{(\ell)}$ at the ℓ-th layer as an input, the output feature matrix $\mathbf{G}^{(\ell+1)}$ of our proposed AGCN model is obtained by applying the following layer-wise propagation rule:

$$\mathbf{G}^{(\ell+1)} = \sigma(\mathbf{G}^{(\ell)}\mathbf{W}^{(\ell)}), \quad \ell = 0, \ldots, L-1, \tag{2.6}$$

which is basically a node embedding transformation that projects the input $\mathbf{G}^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$ into a trainable weight matrix $\mathbf{W}^{(\ell)} \in \mathbb{R}^{F_\ell \times F_{\ell+1}}$ with $F_{\ell+1}$ feature maps, followed by a point-wise non-linear activation function $\sigma(\cdot)$ such as $\text{ReLU}(\cdot) = \max(0, \cdot)$, assuming that $F_{\ell+1} \leq F_\ell \ll N$. It is worth pointing out that after feature aggregation, performing the AGCN layer-wise propagation rule amounts to applying a multi-layer perceptron to the anisotropic feature matrix. In other words, a node latent representation at layer ℓ is transformed linearly via a learned weight matrix to produce the node latent representation at the next layer.

Model prediction. The embedding $\mathbf{G}^{(L)}$ of the last layer of AGCN contains the final output node embeddings, and captures the neighborhood structural information of the graph within L hops. This final node representation can be used as input for downstream tasks such as graph classification, clustering, visualization, link prediction, and node classification. Since the latter task is the focus of this chapter, we apply a softmax classifier as follows:

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{G}^{(L)}\mathbf{W}^{(L)}), \tag{2.7}$$

where $\mathbf{W}^{(L)} \in \mathbb{R}^{F_L \times C}$ is a trainable weight matrix of the last layer, C is the total number of classes, $\text{softmax}(\mathbf{x}) = \exp(\mathbf{x}) / \sum_{c=1}^{C} \exp(\mathbf{x}_c)$ is an activation function applied row-wise, and $\hat{\mathbf{Y}} \in \mathbb{R}^{N \times C}$ is the matrix of predicted labels for graph nodes.

Model complexity. For simplicity, we assume the feature dimensions are the same for all layers, i.e. $F_\ell = F$ for all ℓ, with $F \ll N$. The time complexity of an L-layer AGCN is $\mathcal{O}(L|\mathcal{E}|F + LNF^2)$, where $|\mathcal{E}|$ denotes the number of graph edges. Note that multiplying the normalized adjacency matrix with an embedding costs $\mathcal{O}(|\mathcal{E}|F)$ in time, while multiplying an embedding with a weight matrix costs $\mathcal{O}(NF^2)$. Also, noting that $\text{tr}(\mathbf{H}^{(\ell)\intercal}\tilde{\mathbf{L}}\mathbf{H}^{(\ell)}) = \text{tr}(\tilde{\mathbf{L}}\mathbf{H}^{(\ell)}\mathbf{H}^{(\ell)\intercal})$, it follows that computing this trace operator requires NF^2 scalar multiplications. Hence, the nonlinearity term of AGCN has complexity $\mathcal{O}(NF^2)$.

For memory complexity, an L-layer AGCN requires $\mathcal{O}(LNF + LF^2)$ in memory, where $\mathcal{O}(LNF)$ is for storing all embeddings and $\mathcal{O}(LF^2)$ is for storing all layer-wise weight matrices.

Therefore, the proposed AGCN model has the same time and memory complexity as GCN, while being effective at alleviating the issue of oversmoothing.

Model training. For semi-supervised multi-class classification, the neural network weight parameters are learned by minimizing the cross-entropy loss function

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}_K} \sum_{c=1}^{C} \mathbf{Y}_{ic} \log \hat{\mathbf{Y}}_{ic}, \tag{2.8}$$

over the set $\mathcal{Y}_K$ of all labeled nodes using gradient descent, where $\mathbf{Y}_{ic}$ is equal 1 if node i belongs to class c, and 0 otherwise; and $\hat{\mathbf{Y}}_{ic}$ is the (i, c)-element of the matrix $\hat{\mathbf{Y}}$ from the softmax function, i.e. the probability that the network associates the i-th node with class c.

2.4 Experiments

In this section, we conduct extensive experiments to evaluate the performance of the proposed AGCN framework on several benchmark datasets and carry out a comprehensive comparison with several baseline methods. In all experiments, we consider a two-layer AGCN for semi-supervised node classification

$$\hat{\mathbf{Y}} = \text{softmax}\Big(\text{ReLU}\big(\mathbf{G}^{(0)}\mathbf{W}^{(0)}\big)\mathbf{W}^{(1)}\Big), \tag{2.9}$$

where $\mathbf{W}^{(0)} \in \mathbb{R}^{F\times F_1}$ is a trainable input-to-hidden weight matrix for a hidden layer with F_1 feature maps, $\mathbf{W}^{(1)} \in \mathbb{R}^{F_1\times C}$ is a trainable hidden-to-output weight matrix with C denoting the number of classes, and $\mathbf{G}^{(0)}$ is an $N \times F$ matrix given by

$$\mathbf{G}^{(0)} = \Big(1 - \exp\big(-\beta\,\text{tr}^2(\mathbf{X}^\intercal\tilde{\mathbf{L}}\mathbf{X})\big)\Big)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}. \tag{2.10}$$

Datasets. We demonstrate and analyze the performance of the proposed AGCN model on three citation networks (Cora, Citseer, and Pubmed) and two image datasets (MNIST and CIFAR10). The summary descriptions of these benchmark datasets are as follows:

- Cora is a citation network dataset consisting of 2,708 nodes representing scientific publications and 5,429 edges representing citation links between publications. All publications are classified into 7 classes (research topics). Each node is described by a binary feature vector indicating the absence/presence of the corresponding word from the dictionary, which consists of 1,433 unique words.

- Citeseer is a citation network dataset composed of 3,312 nodes representing scientific publications and 4,723 edges representing citation links between publications. All publications are classified into 6 classes (research topics). Each node is described by a binary feature vector indicating the absence/presence of the corresponding word from the dictionary, which consists of 3,703 unique words.

- Pubmed is a citation network dataset containing 19,717 scientific publications pertaining to diabetes and 44,338 edges representing citation links between publications. All publications are classified into 3 classes. Each node is described by a TF/IDF weighted word vector from the dictionary, which consists of 500 unique words.

- CIFAR10 is an image dataset consisting of 60,000 natural color images, each of which is $32 \times 32 \times 3$ in size and has three color channels (RGB), as shown in Figure 2.2 (left). All images in the dataset are classified into 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 testing images. In our experiments, we randomly select 10,000 images (1,000 images per class) to perform evaluation. For each image, a convolutional neural network is used to extract a feature descriptor, followed by graph construction using the k-nearest neighbors algorithm with $k = 8$.

- MNIST is an image dataset consisting of 70,000 grayscale images of handwritten digits from 0 to 9 (i.e. 10 classes), as shown in Figure 2.2 (right). There are 60,000 training images and 10,000 testing images taken from American Census Bureau employees and American high school students, respectively. We randomly select 1,000 images from each digit for evaluation. Each image is 28×28 in size, and hence it is represented as a 784-dimensional feature vector. The k-nearest neighbors algorithm with $k = 8$ is used to construct the graph for each handwritten digit.

Figure 2.2: Sample images from CIFAR10 (left) and MNIST (right).

Baseline methods. We evaluate the performance of AGCN against several graph-based feature learning models, including DeepWalk [11], Chebyshev networks (ChebyNet) [17], GCN [1], mixture model network (MoNet) [18], graph attention network (GAT) [82], jumping knowledge network (JK-Net) [22], deep graph infomax (DGI) [85], graph wavelet neural network (GWNN) [20], Lanczos network (LanczosNet) [19], graph isomorphism network (GIN) [83], simple graph convolution (SGC) [21], and GCN with pair normalization (GCN-PN) [23]. For baselines, we mainly consider methods that are closely related to AGCN and/or the ones that are state-of-the-art node

classification frameworks. A brief description of these standard baselines can be summarized as follows:

- DeepWalk is a deep learning based framework that learns latent representations of nodes in a graph by leveraging local information obtained from truncated random walks.
- ChebyNet is an efficient spectral-domain graph convolutional neural network that uses recursive Chebyshev polynomial spectral filters to avoid explicit computation of the Laplacian eigenvectors.
- GCN is a semi-supervised graph-based deep learning framework that uses an efficient layer-wise propagation rule that is based on a first-order approximation of spectral graph convolutions.
- MoNet is a spatial-domain graph convolutional neural network that employs a mixture of Gaussian kernels with learnable parameters to model the weight function of pseudo-coordinates, which are associated to the neighboring nodes of each graph node.
- GAT is graph-based neural network architecture that uses an attention mechanism to assign self-attention scores to neighboring node embeddings.
- JK-Net is a graph representation learning approach that learns to selectively exploit information from neighborhoods of differing locality and combines different aggregations at the last layer.
- DGI is graph representation learning framework, which leverages local mutual information maximization across the graph's patch representations to learn node embeddings in an unsupervised manner.
- GWNN is a spectral convolutional neural network, which uses spectral graph wavelets for node feature aggregation.
- LanczosNet is a multiscale graph convolutional network for learning node embedding, which leverages the Lanczos algorithm to construct a low rank approximation of the graph Laplacian for graph convolution.
- GIN is a simple graph neural network architecture, which maps isomorphic graphs to the same representation and non-isomorphic ones to different representations. It is proved to be as powerful as the Weisfeiler-Lehman test for graph isomorphism.

- SGC is a simplified GCN architecture, which consists of a linear feature propagation scheme, followed by multi-class logistic regression.
- GCN-PN is a graph convolutional network, which uses a pair normalization layer to help prevent oversmoothing in deeper graph neural networks.

We also compare our approach with multi-layer perceptron (MLP), manifold regularization (ManiReg) [89], semi-supervised embedding (SemiEmb) [90], LP [91], iterative classification algorithm (ICA) [92], and Planetoid [93].

Implementation details. For fair comparison with prior work, we follow the same experimental setup as [1]. For the CIFAR10 and MNIST image datasets, we randomly select 3,000 images as labeled samples and used the remaining images as unlabeled samples. For unlabeled samples, we select 1,000 images for validation and used the remaining 6,000 images as test samples. All the reported accuracy results of AGCN are averaged over 10 runs with different splits for training, validation and test sets. We train the proposed AGCN model for 200 epochs using Adam optimizer [94] with learning rate 0.01. The training is stopped when the validation loss does not decrease after 10 consecutive epochs. The values of the cross-entropy metric are recorded at the end of each epoch on the training set. The performance comparison between AGCN and GCN over training epochs on the training set is illustrated in Figure 2.3, which shows that the proposed AGCN model yields lower training loss values, indicating higher predictive accuracy. The value of the hyperparameter β is optimized using grid search over the set $\{0, 0.1, 0.2, \ldots, 5\}$.

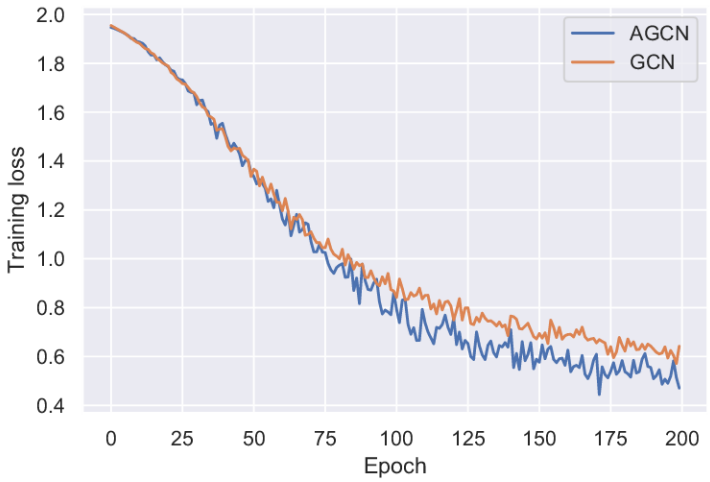


Figure 2.3: Model training history comparison between GCN and proposed AGCN model on the Cora dataset.

2.4.1 Results

The performance of our model is evaluated by conducting a comprehensive comparison with standard baseline methods for node classification using average accuracy as an evaluation metric. The average classification accuracy results in percent are summarized in Table 2.1. Results for baseline methods on the citation networks are taken from the GAT paper [82], and from the corresponding baseline papers for the image datasets. As shown in Table 2.1, our AGCN model outperforms GCN on the citation networks as well as on the image datasets. While DeepWalk does well on the MNIST dataset, it performs poorly on the citation networks compared to AGCN. In addition, AGCN outperforms GAT on the image datasets and the Pubmed citation network, and performs on par with GAT on the Cora dataset. The average accuracy of AGCN on the CIFAR10 dataset is 70%, indicating a performance improvement of 3.4% over GAT. Similarly, AGCN performs on par with DGI on Citeseer, but yields better performance on Cora and Pubmed. Also, AGCN achieves better performance than GWNN, a spectral approach that requires computing an eigendecomposition of the Laplacian, which is usually time- and space-consuming. We can see that AGCN outperforms GIN and LanczosNet across all the citation network datasets, with performance gains of 5.4% and 3.5% on Cora and 5.4% and 3.5% on Citeseer, respectively. Moreover, AGCN performs better than SGC and JK-Net on Cora and Pubmed. Interestingly, despite its simplicity, AGCN achieves a higher accuracy than GCN-NP with improvements of 5.8% and 4% on Citeseer and Cora, respectively. These results demonstrate the significant prediction ability of AGCN in semi-supervised node classification. It is important to mention that both JK-Net and GCN-PN use additional steps such as skip connections and normalization layers to tackle the issue of oversmoothing on graph neural networks, while our proposed AGCN model integrates an oversmoothing prevention term into its neighborhood aggregation scheme using a single step. In other words, the proposed AGCN network is trained in an end-to-end fashion, and eliminates the need to augment deep GNN models with normalization layers or by inserting residual/skip connections between the network's layers in order to improve performance.

Using box plots, Figure 2.4 displays the visual differences in terms of accuracy among AGCN, GAT and GCN on the Cora, Citeseer and Pubmed citation networks. As can be seen, the distribution of the AGCN model has less variability than GCN and GAT on document classification tasks. For instance, the median accuracy score for AGCN on the Pubmed dataset indicates a significant difference in performance between AGCN and the two baseline methods. In addition, the box for AGCN is short, meaning that the accuracy values consistently hover around the average accuracy. However, the box for GAT is taller, implying variable accuracy values compared to AGCN.

Co-training and self-training results. Using the co-training and self-training approaches [84],

Table 2.1: Classification accuracy results on three citations networks and two image datasets. Boldface numbers indicate the best classification performance.

	Average accuracy (%)				
Method	Cora	Citseer	Pubmed	MNIST	CIFAR10
MLP	55.1	46.5	71.4	–	–
ManiReg	59.5	60.1	70.7	94.6	59.7
SemiEmb	59.0	59.6	71.7	–	–
LP	68.0	45.3	63.0	83.4	60.4
DeepWalk	67.2	43.2	65.3	**95.3**	61.3
ICA	75.1	69.1	73.9	–	–
Planetoid	75.7	64.7	77.2	–	–
ChebyNet	81.2	69.8	74.4	–	–
GCN	81.5	70.3	79.0	91.0	61.0
MoNet	81.7	–	78.8	–	–
GAT	**83.0**	72.5	79.0	92.8	66.6
DGI	82.3	71.8	76.8	–	–
GWNN	82.8	71.7	79.1	–	–
LanczosNet	79.5	66.2	78.3	–	–
GIN	77.6	66.1	77.0	–	–
SGC	81.0	71.9	78.9	–	–
JK-Net	82.7	**73.0**	77.9	–	–
GCN-PN	79.0	66.0	78.0	–	–
AGCN	**83.0**	71.8	**79.5**	93.1	**70.0**

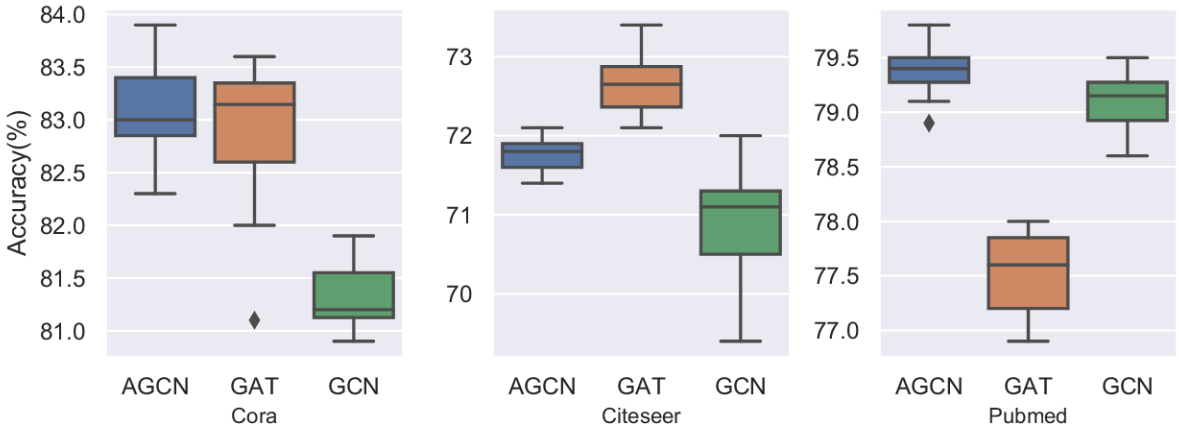


Figure 2.4: Accuracy distributions of AGCN, GAT and GCN on the Cora, Citeseer and Pubmed citation networks.

we compare AGCN with GCN on the Cora, Citeseer and Pubmed datasets with label rates (i.e. proportion of labeled nodes that are used for training) 0.036, 0.052, and 0.003, respectively.

We apply co-training and self-training approaches as well as their intersection and union to train our AGCN model and compare it to GCN. The accuracy results for these four approaches are reported in Figures 2.5, 2.6 and 2.7 on the Cora, Citeseer and Pubmed citation networks, respectively, using training rates of 0.5%, 2% and 4% for Cora and Citseer, and 0.03%, 0.05% and 0.1% for Pubmed. In each bar plot, the bars display the mean and standard error accuracy over 10 runs for both AGCN and GCN using co-training, self-training, union and intersection. In co-training, a partially absorbing random walk is used to find the confidence of node i belongs to class c. The most confident nodes are then added to the training set with label c to train the AGCN model. In self-training, AGCN is applied to find the most confident nodes based on the softmax scores $\hat{\mathbf{Y}} \in \mathbb{R}^{N\times C}$ given by Eq. (2.9). Then, the most confident nodes are added to the labeled set. On the other hand, union and intersection are a combination of co-training and self-training. Union expands the label set with the most confident predictions obtained by random walk and those obtained by AGCN. As can be seen in these figures, our AGCN framework outperforms GCN in most of the cases, particularly for small training sizes. Moreover, notice that the standard deviations are much smaller than the accuracy improvements, indicating that AGCN is robust to random selection of training and test data. Overall, AGCN is consistently the best performing method, delivering robust classification accuracy results.

2.4.2 Statistical Significance Analysis

In this subsection, we conduct statistical significance tests to compare GCN, GAT and AGCN with the objective of selecting the best performing model. More precisely, we apply one-way analysis of variance (ANOVA) to verify whether there is a statistical difference between their mean accuracy scores. ANOVA tests the hypothesis that all group means are equal versus the alternative hypothesis that at least one group is different from the others:

$$\begin{aligned} &H_0 : \mu_1 = \mu_2 = \mu_3 \\ &H_1 : \text{not all group means are equal} \end{aligned} \tag{2.11}$$

where μ_1, μ_2, μ_3 denote the population means for GCN, GAT and AGCN, respectively. While ANOVA is based on the assumption that all sample populations are normally distributed, it is, however, known to be robust to modest violations of the normality assumption.

We perform one-way ANOVA for the accuracy scores data obtained by GCN, GAT and AGCN on the Cora, Citeseer and Pubmed datasets. These results correspond to 10 runs with different splits for training, validation and test sets. As shown in Table 2.2, the small p-values (< 0.05) indicate that differences between accuracy means are statistically significant, where $\alpha = 0.05$

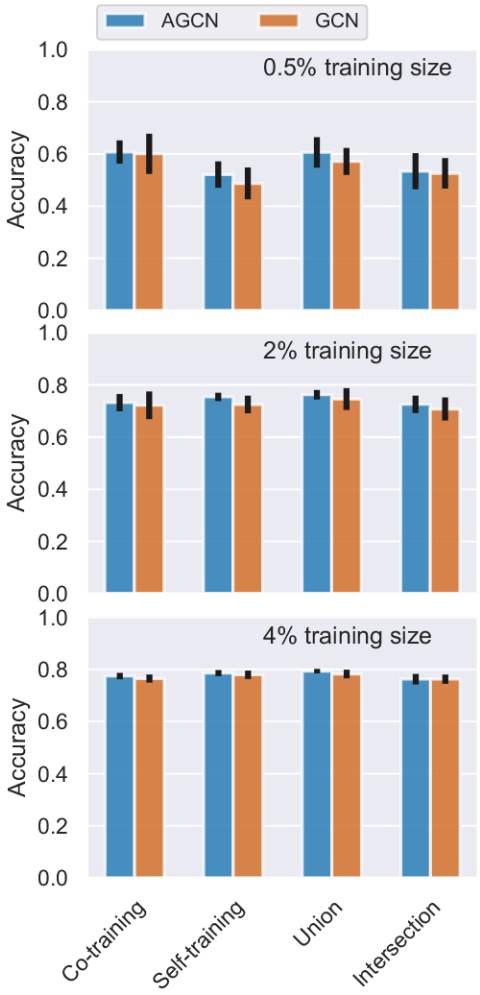


Figure 2.5: Classification accuracy of AGCN compared to GCN for different training set sizes on the Cora dataset.

is the significance level. A significance level of 0.05 indicates a 5% risk of concluding that a difference exists when there is no actual difference.

Table 2.2: One-way ANOVA p-values for the accuracy scores data obtained by AGCN, GCN and GAT on the Cora, Citeseer and Pubmed datasets.

	Cora	Citeseer	Pubmed
p-value	1.0×10^{-3}	2.51×10^{-7}	1.21×10^{-6}

Since our ANOVA analysis shows an overall statistically significant difference in group means, we now need to determine which specific groups (compared with each other) are different in terms of mean accuracies by performing multiple pairwise comparison (post-hoc comparison) analysis using Tukey's test, which compares all possible pairs of means. The pairwise multiple comparison

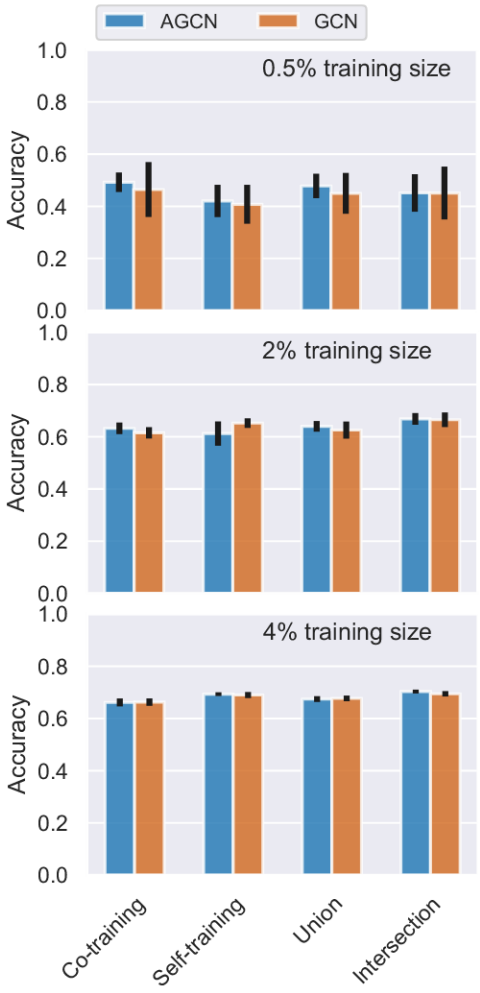


Figure 2.6: Classification accuracy of AGCN compared to GCN for different training set sizes on the Citeseer dataset.

results of GCN, GAT, and AGCN methods using Tukey's test on the Cora, Citeseer and Pubmed datasets are shown in Figures 2.8, 2.9 and 2.10, respectively. The table above each figure reports the results of the multiple comparison of means. In the case of the Cora dataset, for instance, we can see from the result shown in the table of Figure 2.8 that we reject the hypotheses that "AGCN and GCN" and "GAT and GCN" have the same mean, but we fail to reject the "AGCN and GAT" pair and conclude that they have equal mean accuracies. In the meandiff column, the difference of accuracy means between related groups is reported, followed by lower and upper limits for 95% confidence intervals for the true mean difference. In the reject column, "True" indicates that there is significant evidence to reject the null hypothesis at the given significance level, i.e. there is a significant statistical difference between the means. The results from Tukey's test reported in the tables of Figures 2.9 and 2.10 show that we reject the hypotheses that "AGCN and GAT", "AGCN and GCN" and "GAT and GCN" have the same mean, indicating statistical significant differences.

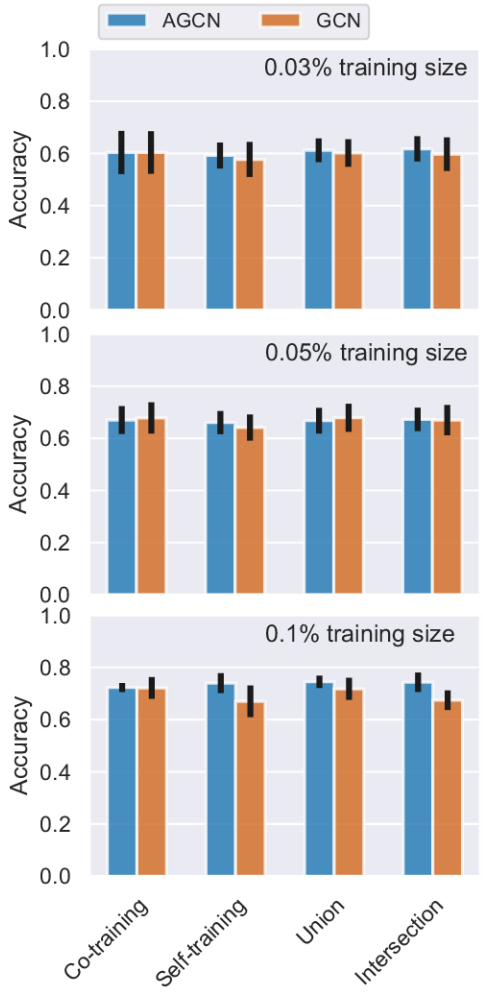


Figure 2.7: Classification accuracy of AGCN compared to GCN for different training set sizes on the Pubmed dataset.

In each figure, the 95% confidence intervals plots are displayed as horizontal bars. The blue bar in Figure 2.8 shows the comparison interval for the AGCN mean accuracy, which does not overlap with the comparison interval for the GCN mean accuracy, shown in red. The comparison interval for the GAT mean accuracy, shown in gray, overlaps with the comparison interval for the AGCN mean accuracy. Hence, the accuracy means for AGCN and GAT are not significantly different from each other on the Cora dataset, meaning that both methods performs on par with each other.

In Figure 2.9, the blue bar shows the comparison interval for the AGCN mean accuracy, which does not overlap with the comparison intervals for the GCN and GAT mean accuracies, shown in red. The disjoint comparison intervals indicate that the group means are significantly different from each other. Similarly, the blue bar in Figure 2.10 shows the comparison interval for the AGCN mean accuracy, which does not overlap with the comparison intervals for the GCN and GAT mean accuracies, shown in red. Hence, we conclude that AGCN is the best performing method on the

Pubmed dataset. This visual observation is consistent with the results reported in Table 2.1.

group1	group2	meandiff	lower	upper	reject
AGCN	GAT	-0.0004	-0.00472	0.00392	False
AGCN	GCN	-0.0079	-0.01222	-0.00358	True
GAT	GCN	-0.0075	-0.01182	-0.00318	True

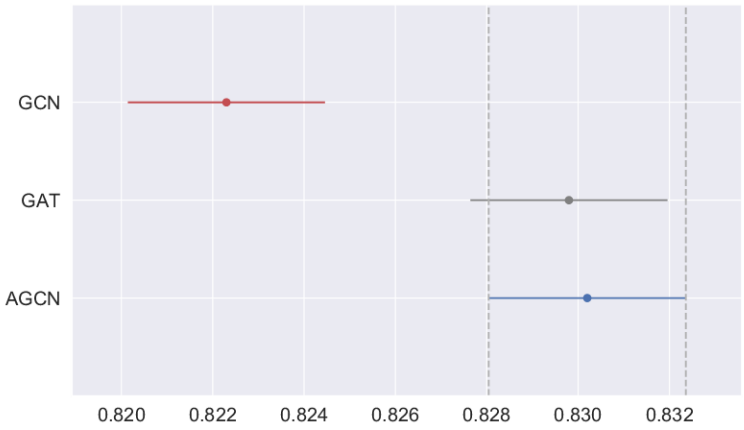


Figure 2.8: Pairwise multiple comparison between AGCN, GAT, and GCN methods using Tukey's test on the Cora dataset.

group1	group2	meandiff	lower	upper	reject
AGCN	GAT	0.0078	0.00211	0.01349	True
AGCN	GCN	-0.0094	-0.01509	-0.00371	True
GAT	GCN	-0.0172	-0.02289	-0.01151	True

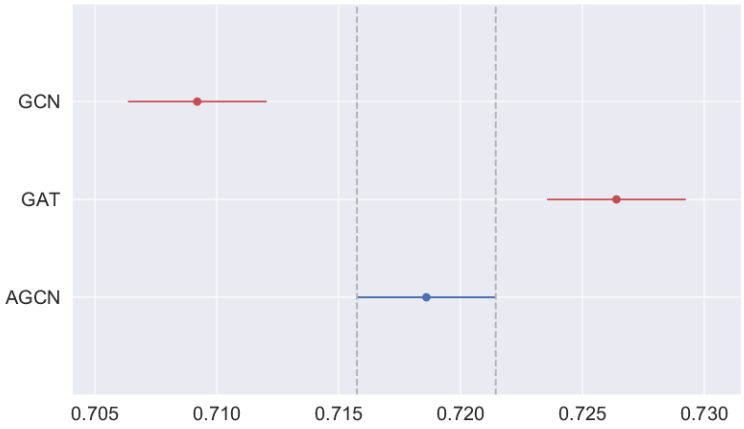


Figure 2.9: Pairwise multiple comparison between AGCN, GAT, and GCN methods using Tukey's test on the Citeseer dataset.

group1	group2	meandiff	lower	upper	reject
AGCN	GAT	-0.009	-0.01193	-0.00607	True
AGCN	GCN	-0.0048	-0.00773	-0.00187	True
GAT	GCN	0.0042	0.00127	0.00713	True

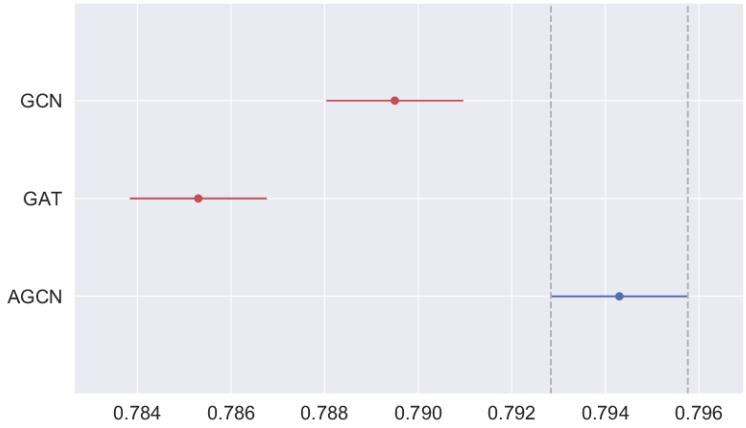


Figure 2.10: Pairwise multiple comparison between AGCN, GAT, and GCN methods using Tukey's test on the Pubmed dataset.

2.4.3 Visualization

The feature embeddings learned by AGCN can be visualized using the t-Distributed Stochastic Neighbor Embedding (t-SNE) [95], which is a dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a two- or three-dimensional space. Figure 2.11 displays the t-SNE embeddings of the output embeddings by the first convolutional layer of AGCN (top) and GCN (bottom) on the MNIST dataset. As can be seen, the two-dimensional embeddings corresponding to AGCN are more separable than the ones corresponding to GCN. With GCN features, the points are not discriminated very well, while with AGCN features the points are discriminated much better and clearly show the clusters corresponding to the ten digit labels of the MNIST dataset. Hence, AGNC learns more discriminative features for node classification tasks, indicating the superior performance of anisotropic diffusion over linear diffusion. Moreover, Figure 2.11 shows that the AGCN approach is exploratory in nature in the sense that it can discover patterns and meaningful sub-groups in a dataset.

2.4.4 Robustness to Oversmoothing

In order to assess robustness to oversmoothing, we study the performance variation for our multi-layer model on the Cora dataset with respect to the number of layers. Figure 2.12 shows how the

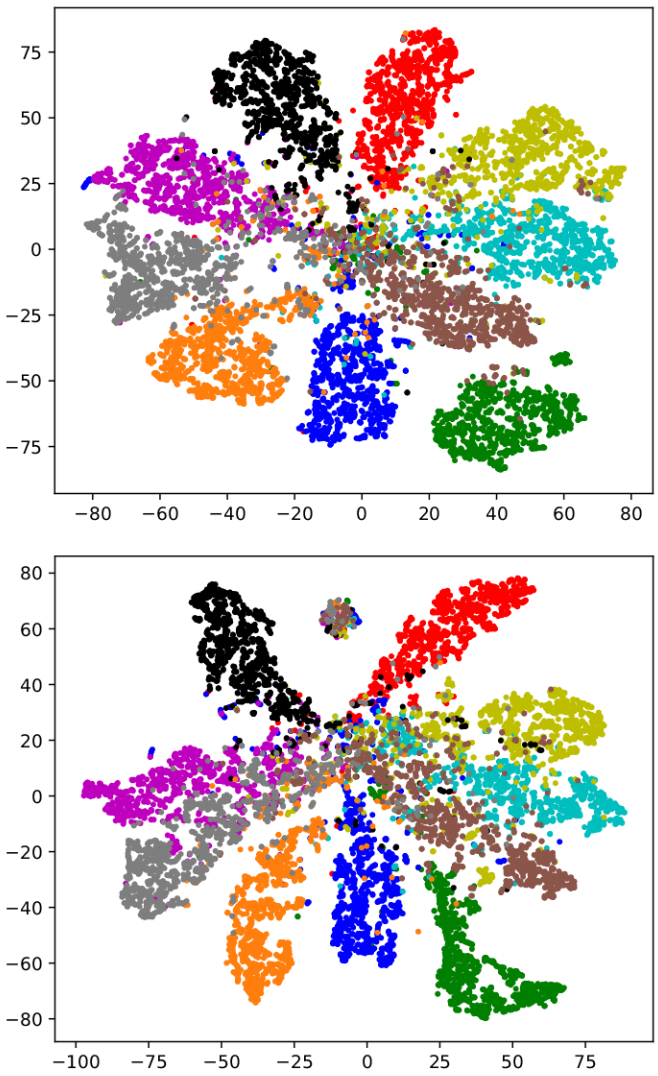


Figure 2.11: t-SNE feature visualization of the output embeddings by the first convolutional layer of AGCN (top) and GCN (bottom), respectively, on the MNIST dataset. Each color denotes a class.

node classification accuracy changes with the network's depth. As can be seen, there is a sharp drop in GCN's accuracy when the number of layers is larger than 4, while AGCN's performance does not significantly degrade as the number of layers increases. Note that the performance gap between AGCN and GCN substantially increases when the network's depth is beyond 4, indicating that AGCN is more robust to oversmoothing. It is worth pointing out that the objective of our proposed anisotropic convolution is to mitigate the oversmoothing issue that causes drops in classification performance for multi-layer GCNs, and not to show that the deeper AGCN, the better. Also, increasing the depth of the network leads to increased number of parameters, causing overfitting and hence resulting in performance drop. As shown in Figure 2.12, a 6-layer AGCN yields

an accuracy of 80%, outperforming several baselines including LanczosNet, GIN and GCN-PN. The latter baseline is designed specifically for tackling the issue of oversmoothing in GCNs by employing a normalization layer after each convolutional layer.

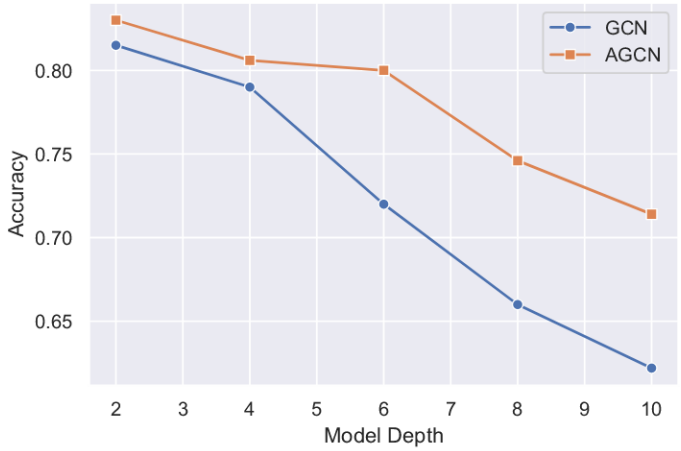


Figure 2.12: Performance comparison between AGCN and GCN on the Cora dataset as we increase the number of layers.

2.4.5 Parameter Sensitivity Analysis

For each benchmark dataset used in the experiments, we test the performance of AGCN using different values for the hyper-parameter β of the anisotropic diffusion term. In the case of the Pubmed dataset, for instance, the effect of β on the performance of AGCN is illustrated in Figure 2.13, which shows the accuracy of AGCN along with the standard error for different values of β. As can be seen, $\beta = 0.4$ yields the best value for the AGCN accuracy on the Pubmed dataset.

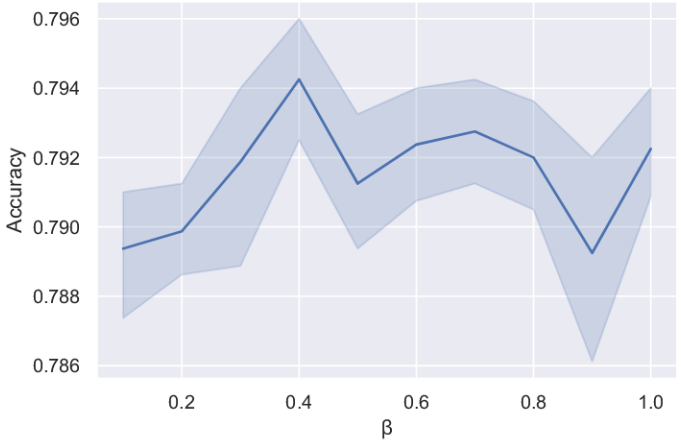


Figure 2.13: AGCN accuracy results for different values of β on the Pubmed dataset.

2.4.6 Discussion

While the proposed AGCN model shows promising results for end-to-end learning on graphs and mitigates the issue of oversmoothing, its performance is, however, tied to optimizing via grid search with cross-validation the hyper-parameter of the anisotropic diffusion term for each dataset. Another shortcoming of AGCN is that it only takes into account immediate neighbors. This limitation can be circumvented through higher-order message passing by leveraging multi-hop neighbors using powers of the adjacency matrix and hence aggregating learned node representations from both immediate and distant neighbors.

CHAPTER 3

Graph Fairing Convolutional Networks for Anomaly Detection

Graph convolution is a fundamental building block for many deep neural networks on graph-structured data. In this chapter, we introduce a simple, yet very effective graph convolutional network with skip connections for semi-supervised anomaly detection. The proposed layerwise propagation rule of our model is theoretically motivated by the concept of implicit fairing in geometry processing, and comprises a graph convolution module for aggregating information from immediate node neighbors and a skip connection module for combining layer-wise neighborhood representations. This propagation rule is derived from the iterative solution of the implicit fairing equation via the Jacobi method. In addition to capturing information from distant graph nodes through skip connections between the network's layers, our approach exploits both the graph structure and node features for learning discriminative node representations. These skip connections are integrated by design in our proposed network architecture. The effectiveness of our model is demonstrated through extensive experiments on five benchmark datasets, achieving better or comparable anomaly detection results against strong baseline methods. We also demonstrate through an ablation study that skip connection helps improve the model performance.

3.1 Introduction

Anomaly detection is of paramount importance when deploying artificial intelligence systems, which often encounter unexpected or abnormal items/events that deviate significantly from the

majority of data. Anomaly detection techniques are widely used in a variety of real-world applications, including, but not limited to, intrusion detection, fraud detection, healthcare, Internet of Things, Industry 4.0 and beyond, surveillance, and social networks [25, 96, 97]. Most of these techniques often involve a training set where no anomalous instances are encountered, and the challenge is to identify suspicious items or events, even in the absence of abnormal instances. In practical settings, the output of an anomaly detection model is usually an alert that triggers every time there is a anomaly or a pattern in the data that is atypical.

Detecting anomalies is a challenging task, primarily because the anomalous instances are not known a priori and also the vast majority of observations in the training set are normal instances. Therefore, the mainstream approach in anomaly detection has been to separate the normal instances from the anomalous ones by using unsupervised learning models. One-class support vector machines (OC-SVM) are a classic example [98], which is a one-class classification model trained on data that has only one class (i.e. normal class) by learning a discriminative hyperplane boundary around the normal instances. Another commonly-used anomaly detection approach is support vector data description (SVDD) [24], which basically finds the smallest possible hypersphere that contains all instances, allowing some instances to be excluded as anomalies. Zhang *et al.* [99] presented a graph model-based multiscale feature fitting method for unsupervised anomaly detection and localization. Arias *et al.* [100] introduced an unsupervised parameter-free analytic isolation and distance-based anomaly detection algorithm, which integrates both distance and isolation metrics. However, these approaches rely on hand-crafted features, are unable to appropriately handle high-dimensional data, and often suffer from computational scalability issues.

Deep learning has recently emerged as a very powerful way to hierarchically learn abstract patterns from data, and has been successfully applied to anomaly detection, showing promising results in comparison with shallow methods [101]. Ruff *et al.* [26] extend the shallow one-class classification SVDD approach to the deep learning setting by proposing a deep learning based SVDD framework for anomaly detection using an anomaly detection based objective. Deep SVDD is an unsupervised learning model that learns to extract the common factors of variation of the data distribution by training a neural network while minimizing the volume of a hypersphere that encloses the network representations of the data. Also, Ruff *et al.* [27] introduce a deep semi-supervised anomaly detection (Deep SAD) approach, which is a generalization of the unsupervised Deep SVDD technique to the semi-supervised setting. Deep SAD differs from Deep SVDD in that its objective function also includes a loss term for labeled normal and anomalous instances. The more diverse the labeled anomalous instances in the training set, the better the anomaly detection performance of Deep SAD. The key difference between deep one-class models such as Deep SVVD and

semi-supervised anomaly detection methods such as Deep SAD lies in the way they are trained and the amount of labeled data required. The former is an unsupervised anomaly detection method that requires only normal data during training. It is trained to learn a representation of normal data, which is then used to distinguish between normal and anomalous data. Semi-supervised anomaly detection, on the other hand, is a hybrid approach that generally uses both normal and anomalous data during training. The model is trained using both labeled and unlabeled data, where the labeled data consists of a small portion of anomalous data and a large portion of normal data. The main advantage of semi-supervised anomaly detection is that it can achieve higher accuracy than unsupervised methods, as the use of labeled data during training provides additional information, enabling the model to better distinguish between normal and anomalous data points.

Owing to the recent developments in deep semi-supervised learning on graph-structured data, there has been a surge of interest in the adoption of graph neural networks for learning latent representations of graphs [1, 17]. Defferrard *et al.* [17] introduce the Chebyshev network, an efficient spectral-domain graph convolutional neural network that uses recursive Chebyshev polynomial spectral filters to avoid explicit computation of the Laplacian spectrum. These filters are localized in space, and the learned weights can be shared across different locations in a graph. An efficient variant of graph neural networks is graph convolutional networks (GCNs) [1], which is an upsurging semi-supervised graph-based deep learning framework that uses an efficient layer-wise propagation rule based on a first-order approximation of spectral graph convolutions. The feature vector of each graph node in GCN is updated by essentially applying a weighted sum of the features of its immediate neighboring nodes. While significant strides have been made in addressing anomaly detection on graph-structured data [102], it still remains a daunting task on graphs due to various challenges, including graph sparsity, data nonlinearity, and complex modality interactions [28]. Ding *et al.* [28] design a GCN-based autoencoder for anomaly detection on attributed networks by taking into account both topological structure and nodal attributes. The encoder of this unsupervised approach encodes the attribute information using the output GCN embedding, while the decoder reconstructs both the structure and attribute information using non-linearly transformed embeddings of the output GCN layer. The basic idea behind anomaly detection methods based on reconstruction errors is that the normal instances can be reconstructed with small errors, while anomalous instances are often reconstructed with large errors. More recently, Kumagai *et al.* [29] have proposed two GCN-based models for semi-supervised anomaly detection. The first model uses only labeled normal instances, whereas the second one employs labeled normal and anomalous instances. Both models are trained to minimize the volume of a hypersphere that encloses the GCN-learned node embeddings of normal instances, while embedding the anomalous

ones outside the hypersphere.

Inspired by the implicit fairing concept in geometry processing for triangular mesh smoothing [2], we introduce a graph fairing convolutional network architecture, which we call GFCN, for deep semi-supervised anomaly detection. In addition to performing graph convolution, GFCN uses a skip connection to combine both the initial node representation and the aggregated node neighborhood representation, enabling it to memorize information across distant nodes. While most graph convolutions with skip connections are based on heuristics, GFCN is theoretically motivated by implicit fairing and derived from the Jacobi iterative method. In contrast to GCN-based methods that use a first-order approximation of spectral graph convolutions and a renormalization trick in their layer-wise propagation rules to avoid numerical instability, our GFCN model does not require any renormalization, while still maintaining the key property of convolution as a neighborhood aggregation operator. Hence, repeated application of the GFCN's layer-wise propagation rule provides a computationally efficient convolutional process, leading to numerical stability while avoiding the issue of exploding/vanishing gradients. The proposed framework achieves better anomaly detection performance, as GFCN uses a multi-layer architecture, together with skip connections, and non-linear activation functions to extract high-order information of graphs as discriminative features. Multi-layer architectures enable the model to learn hierarchical representations of the graph, where lower layers capture lower-level features and higher layers capture higher-level abstractions. Skip connections allow information to bypass intermediate layers and preserve low-level details, improving the flow of information and preventing vanishing gradients, and more importantly leading to more accurate representations, thereby yielding a more effective detection of anomalies. Moreover, GFCN inherits all benefits of GCNs, including accuracy, efficiency and ease of training.

In addition to capturing information from distant graph nodes through skip connections between layers, the proposed GFCN model is flexible and exploits both the graph structure and node features for learning discriminative node representations in an effort to detect anomalies in a semi-supervised setting. Not only does GFCN outperforms strong anomaly detection baselines, but it is also surprisingly simple, yet very effective at identifying anomalies. The main contributions of this work can be summarized as follows:

- We propose a novel multi-layer graph convolutional network with a skip connection for semi-supervised anomaly detection by effectively exploiting both the graph structure and attribute information.
- We introduce a learnable skip-connection module, which helps nodes propagate through the network's layers and hence substantially improves the quality of the learned node represen-

tations.

- We analyze the complexity of the proposed model and train it on a regularized, weighted cross-entropy loss function by leveraging unlabeled instances to improve performance.
- We demonstrate through extensive experiments that our model can capture the anomalous behavior of graph nodes, leading to state-of-the-art performance across several benchmark datasets.

The rest of this chapter is organized as follows. In Section 2, we review important relevant work. In Section 3, we outline the background for spectral graph theory and present the problem formulation. In Section 4, we introduce a graph convolutional network architecture with skip connection for deep semi-supervised anomaly detection. In Section 5, we present experimental results to demonstrate the competitive performance of our approach on five standard benchmarks. Finally, we conclude in Section 6 and point out future work directions.

3.2 Related Work

The basic goal of anomaly detection is to identify abnormal instances, which do not conform to the expected pattern of other instances in a dataset. To achieve this goal, various anomaly detection techniques have been proposed, which can distinguish between normal and anomalous instances [96]. Most mainstream approaches are one-class classification models [24, 98] or graph-based anomaly methods [102].

Deep Learning for Anomaly Detection. While shallow methods such as one-class classification models require explicit hand-crafted features, much of the recent work in anomaly detection leverages deep learning [25], which has shown remarkable capabilities in learning discriminative feature representations by extracting high-level features from data using multilayered neural networks. Ruff *et al.* [26] develop a deep SVDD anomaly detection framework, which is basically an extension of the shallow one-class classification SVDD approach. The basic idea behind SVDD is to find the smallest hypersphere that contains all instances, except for some anomalies. Deep SVDD is an unsupervised learning model that learns to extract the common factors of variation of the data distribution by training a neural network while minimizing the volume of a hypersphere that encloses the network representations of the data. The centroid of the hypersphere is usually set to the mean of the feature representations learned by performing a single initial forward pass. In order to improve model performance, Ruff *et al.* [27] propose Deep SAD, a generalization of the unsupervised Deep SVDD to the semi-supervised setting. The key difference between these two deep

anomaly detection models is that the objective function of Deep SAD also includes a loss term for labeled normal and anomalous instances. The idea behind this loss term is to minimize (resp. maximize) the squared Euclidean distance between the labeled normal (resp. anomalous) instances and the hypersphere centroid. However, both Deep SVDD and Deep SAD suffer from the hypersphere collapse problem due to the learning of a trivial solution. In other words, the network's learned features tend to converge to the centroid of the hypersphere if no constraints are imposed on the architectures of the models. Cevikalp *et al.* [103] considered the hypersphere centers as parameters that can be learned and updated according to the evolving deep feature representations. Another line of work uses deep generative models to address the anomaly detection problem [104, 105]. These generative networks are able to localize anomalies, particularly in images, by simultaneously training a generator and a discriminator, enabling the detection of anomalies on unseen data based on unsupervised training of the model on anomaly-free data [106]. However, the use of deep generative models in anomaly detection has been shown to be quite problematic and unintuitive, particularly on image data [107].

Graph Convolutional Networks for Anomaly Detection. GCNs have recently become the de facto model for learning representations on graphs, achieving state-of-the-art performance in various application domains, including anomaly detection [28, 29]. Ding *et al.* [28] present an unsupervised graph anomaly detection framework using a GCN-based autoencoder. This approach leverages both the topological structure and nodal attributes, with an encoder that maps the attribute information into a low-dimensional feature space and a decoder that reconstructs the structure as well as the attribute information using the learned latent representations. The basic idea behind this GCN-based autoencoder is that the normal instances can be reconstructed with small errors, while anomalous instances are often reconstructed with large errors. However, methods based on reconstruction errors are prone to outliers and often require noise-free data for training. On the other hand, some of the main challenges associated with graph anomaly detection is the lack of labeled graph nodes (i.e. no information is available about which instances are actually anomalous and which ones are normal) and data imbalance, as abnormalities occur rarely and hence a tiny fraction of instances is expected to be anomalous. To circumvent these issues, Kumagai *et al.* [29] propose two semi-supervised anomaly detection models using GCNs for learning latent representations. The first model uses only labeled normal instances, while the second one employs both labeled normal and anomalous instances. However, both models are trained to minimize the volume of a hypersphere that encloses the GCN-learned node embeddings of normal instances, and hence they also suffer from the hypersphere collapse problem. By contrast, our semi-supervised GFCN model does not suffer from the above mentioned issues. In addition to leveraging the graph structure

and node attributes, GFCN learns from both labeled and unlabeled data in order to improve model performance.

Graph Neural Networks with Skip Connections. Despite the success of GNN-based models in learning node representations, they are prone to over-smoothing, which can negatively impact their performance. Over-smoothing occurs when stacking multiple graph convolution layers causes node representations to become indistinguishable, leading to a loss of valuable information. To tackle the over-smoothing problem, several approaches that leverage skip connections have been proposed. Skip connections can be categorized into four main types: residual connections, initial connections, jumping connections, and dense connections [108]. JK-Net [22] uses jumping knowledge network connections to connect each layer to the last one, maintaining the feature mappings in lower layers. APPNP [30], which approximate PageRank with power iteration, uses initial connection by connecting each layer to the original feature matrix. By decoupling feature transformation and propagation, APPNP can aggregate information from multi-hop neighbors without increasing the number of layers in the network. GCNII [32] employs initial residual and identity mapping to mitigate the over-smoothing problem. At each layer, the initial residual constructs a skip connection from the input layer, while the identity mapping adds an identity matrix to the weight matrix. ResGCN [31] is a residual graph convolutional network that extends the depth of GCNs by using residual/dense connections and dilated convolutions. In our proposed GFCN model, we apply a skip connection that reuses the initial node features at each layer with the goal of combining both the aggregated node neighborhood representation and the initial node representation. While most graph convolutions with skip connections are based on heuristics, our GFCN model is theoretically motivated by implicit fairing and its layerwise propagation rule is derived from the iterative solution of the implicit fairing equation via the Jacobi method.

3.3 Preliminaries and Problem Statement

We introduce our notation and present a brief background on spectral graph theory, followed by our problem formulation of semi-supervised anomaly detection on graphs.

Basic Notions. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of N nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. The graph structure is encoded by an $N \times N$ adjacency matrix $\mathbf{A} = (\mathbf{A}_{ij})$ whose (i, j)-th entry is equal to the weight of the edge between neighboring nodes i and j, and 0 otherwise. We also denote by $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^\intercal$ an $N \times F$ feature matrix of node attributes, where $\mathbf{x}_i$ is an F-dimensional row vector for node i. This real-valued feature vector is often referred to as a graph signal, which assigns a value to each node in the graph.

Spectral Graph Theory. The normalized Laplacian matrix is defined as

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}, \tag{3.1}$$

where $\mathbf{D} = \mathrm{diag}(\mathbf{A1})$ is the diagonal degree matrix, and $\mathbf{1}$ is an N-dimensional vector of all ones. Since the normalized Laplacian matrix is symmetric positive semi-definite, it admits an eigendecomposition given by $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\intercal}$, where $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_N)$ is an orthonormal matrix whose columns constitute an orthonormal basis of eigenvectors and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ is a diagonal matrix comprised of the corresponding eigenvalues such that $0 = \lambda_1 \leq \cdots \leq \lambda_N \leq 2$. If $\mathcal{G}$ is a bipartite graph, then the spectral radius (i.e. largest absolute value of all eigenvalues) of the normalized Laplacian matrix is equal to 2. The normalized Laplacian matrix has eigenvalues in the range [0,2], which makes spectral graph analysis algorithms more stable and reliable compared to algorithms that use the unnormalized Laplacian matrix with eigenvalues that can be much larger. Moreover, scaling by the inverse square root of the degree matrix helps reduce the influence of highly connected nodes.

Problem Statement. Anomaly detection aims at identifying anomalous instances, which do not conform to the expected pattern of other instances in a dataset. It differs from binary classification in that it distinguishes between normal and anomalous observations. Also, the distribution of anomalies is not usually known a priori.

Let $\mathcal{D}_l = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_l}$ be a set of labeled data points $\mathbf{x}_i \in \mathbb{R}^F$ and their associated known labels $y_i \in \{0, 1\}$ with 0 and 1 representing "normal" and "anomalous" observations, respectively, and $\mathcal{D}_u = \{\mathbf{x}_i\}_{i=N_l+1}^{N_l+N_u}$ be a set of unlabeled data points, where $N_l + N_u = N$. Hence, each node i can be labeled with a 2-dimensional one-hot encoding vector $\mathbf{y}_i = (y_i, 1 - y_i)$.

The goal of semi-supervised anomaly detection on graphs is to estimate the anomaly scores of the unlabeled graph nodes. Nodes with high anomaly scores are considered anomalous, while nodes with lower scores are deemed normal.

3.4 Proposed Method

In this section, we begin by succinctly summarizing some of the most common spectral filters on graphs. Then, we propose a graph convolutional network with skip connection using the concept of implicit fairing on graphs. In particular, we examine the main components of the proposed architecture and analyze the complexity of the model. In addition, we introduce an anomaly scoring function defined in terms of the weighted cross-entropy between the ground-truth labels of the graph test nodes and the model's predicted probabilities.

3.4.1 Spectral Graph Filtering

The idea of spectral filtering on graphs was first introduced in [109] in the context of triangular mesh smoothing. The goal of spectral graph filtering is to use polynomial or rational polynomial filters defined as functions of the graph Laplacian (or equivalently its eigenvalues) in an effort to attenuate high-frequency noise corrupting the graph signal. These functions are usually referred to as frequency responses or transfer functions. While polynomial filters have finite impulse responses, their rational counterparts have infinite impulse responses. Despite the fact that the Laplacian matrix is commonly used in spectral graph theory, it does not, however, provide a natural way to normalize the frequency domain representation of a graph signal, which can lead to scaling and convergence issues in spectral graph filtering. In contrast, the normalized Laplacian matrix provides a way to normalize the frequency domain representation of a graph signal, which can improve the stability and convergence properties of spectral graph filtering. Specifically, the normalized Laplacian matrix is scaled by the inverse square root of the degree matrix, which helps normalize the contributions of each node's neighbors to the overall graph signal. Applying a spectral graph filter with transfer function h on the graph signal $\mathbf{X} \in \mathbb{R}^{N \times F}$ yields

$$\mathbf{H} = h(\mathbf{L})\mathbf{X} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^\intercal\mathbf{X} = \mathbf{U}\,\mathrm{diag}(h(\lambda_i))\mathbf{U}^\intercal\mathbf{X}, \tag{3.2}$$

where $\mathbf{H}$ is the filtered graph signal. However, this filtering process necessitates the computation of the eigenvalues and eigenvectors of the Laplacian matrix, which is prohibitively expensive for large graphs. To circumvent this issue, spectral graph filters are usually approximated using Chebyshev polynomials [17, 110, 111] or rational polynomials [112–114].

3.4.2 Implicit Fairing

Graph fairing refers to the process of designing and computing smooth graph signals on a graph in order to filter out undesirable high-frequency noise while retaining the graph geometric features as much as possible. The implicit fairing method, which uses implicit integration of a diffusion process for graph filtering, has shown to allow for both efficiency and stability [2]. The implicit fairing filter is an infinite impulse response filter whose transfer function is given by $h_s(\lambda) = 1/(1+s\lambda)$, where s is a positive parameter. Hence, performing graph filtering with implicit fairing is equivalent to solving the following sparse linear system:

$$(\mathbf{I} + s\mathbf{L})\mathbf{H} = \mathbf{X}, \tag{3.3}$$

which we refer to as implicit fairing equation. It is worth pointing out that this equation can also be obtained by minimizing the following objective function

$$\mathcal{J}(\mathbf{H}) = \frac{1}{2}\|\mathbf{H} - \mathbf{X}\|_F^2 + \frac{s}{2}\,\mathrm{tr}(\mathbf{H}^\intercal \mathbf{L}\mathbf{H}), \tag{3.4}$$

where $\|\cdot\|_F$ and $\mathrm{tr}(\cdot)$ denote the Frobenius norm and trace operator, respectively.

The implicit fairing filter enjoys several nice properties, including unconditional stability as $h_s(\lambda)$ is always in $[0, 1]$, and also preservation of the average value (i.e. DC value or centroid) of the graph signal as $h_s(0) = 1$ for all s. As shown in Figure 3.1, the higher the value of the scaling parameter, the closer the implicit fairing filter becomes to the ideal low-pass filter.

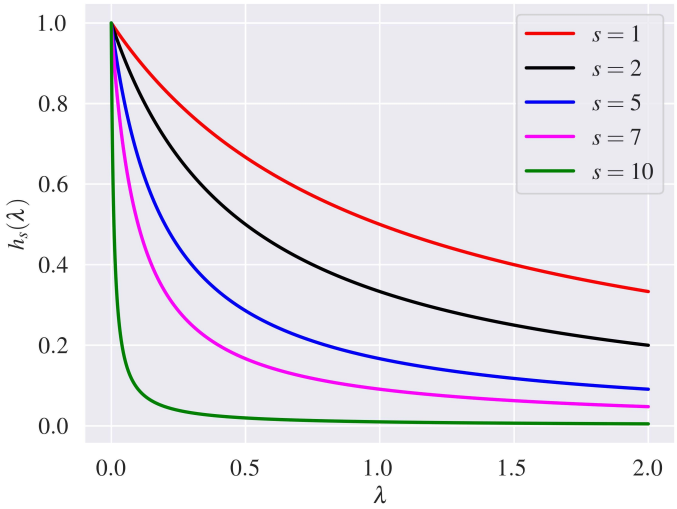


Figure 3.1: Transfer function of the implicit fairing filter for various values of the scaling parameter.

3.4.3 Spectral Analysis

The matrix $\mathbf{I} + s\mathbf{L}$ is symmetric positive definite with minimal eigenvalue equal to 1 and maximal eigenvalue bounded from above by $1 + 2s$. Hence, the condition number κ of $\mathbf{I} + s\mathbf{L}$ satisfies

$$\kappa \leq 1 + 2s, \tag{3.5}$$

where κ, which is defined as the ratio of the maximum to minimum stretching, is also equal to the maximal eigenvalue of $\mathbf{I} + s\mathbf{L}$. Intuitively, the condition number measures how much can a change (i.e. small perturbation) in the right-hand side of the implicit fairing equation affects the solution. In fact, it can be readily shown that the resulting relative change in the solution of the implicit fairing equation is bounded from above by the condition number multiplied by the relative change in the right-hand side.

3.4.4 Iterative Solution

One of the simplest iterative techniques for solving a matrix equation is Jacobi's method, which uses matrix splitting. Since the matrix $\mathbf{I} + s\mathbf{L}$ can be split into the sum of a diagonal matrix and an off-diagonal matrix

$$\mathbf{I} + s\mathbf{L} = (1+s)\mathbf{I} - s\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \tag{3.6}$$

the implicit fairing equation can then be solved iteratively using the Jacobi method as follows:

$$\mathbf{H}^{(t+1)} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{H}^{(t)}\boldsymbol{\Theta}_s + \mathbf{X}\widetilde{\boldsymbol{\Theta}}_s, \tag{3.7}$$

where $\boldsymbol{\Theta}_s = \mathrm{diag}(s/(1+s))$ and $\widetilde{\boldsymbol{\Theta}}_s = \mathrm{diag}(1/(1+s))$ are $F \times F$ diagonal matrices, each of which has equal diagonal entries, and $\mathbf{H}^{(t)}$ is the t-th iteration of $\mathbf{H}$. Since the spectral radius of the normalized adjacency matrix is equal to 1, it follows that the spectral radius of the Jacobi's iteration matrix

$$\mathbf{C} = \frac{s}{1+s}\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \tag{3.8}$$

is equal to $s/(1+s)$, which is always smaller than 1. Hence, the convergence of the iterative method given by Eq. (3.7) holds.

3.4.5 Graph Fairing Convolutional Network

At the core of graph representation learning is the concept of propagation rule, which determines how information is passed between nodes in a graph. It involves updating the current node features by aggregating information from their neighboring nodes, followed by a non-linear activation function to produce an updated representation for the node. Inspired by the Jacobi iterative solution of the implicit fairing equation, we propose a multi-layer graph fairing convolutional network (GFCN) with the following layer-wise propagation rule:

$$\mathbf{H}^{(\ell+1)} = \sigma(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{H}^{(\ell)}\boldsymbol{\Theta}^{(\ell)} + \mathbf{X}\widetilde{\boldsymbol{\Theta}}^{(\ell)}), \tag{3.9}$$

where $\boldsymbol{\Theta}^{(\ell)}$ and $\widetilde{\boldsymbol{\Theta}}^{(\ell)}$ are learnable weight matrices, $\sigma(\cdot)$ is an element-wise activation function, $\mathbf{H}^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$ is the input feature matrix of the ℓ-th layer with F_ℓ feature maps for $\ell = 0, \ldots, L-1$. The input of the first layer is the initial feature matrix $\mathbf{H}^{(0)} = \mathbf{X}$.

Note that in addition to performing graph convolution, which essentially averages the features of the immediate (i.e. first-order or 1-hop) neighbors of nodes, the layer-wise propagation rule of GFCN also applies a skip connection that reuses the initial node features, as illustrated in Figure 3.2. In other words, GFCN combines both the aggregated node neighborhood representation and the initial node representation, hence memorizing information across distant nodes. While

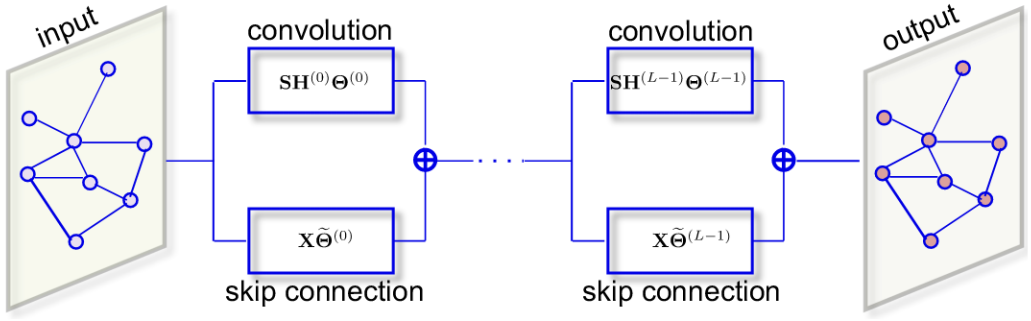


Figure 3.2: Schematic layout of the proposed GFCN architecture. Each block comprises a graph convolution and a skip connection, followed by an activation function, where $\mathbf{S}$ denotes the normalized adjacency matrix. The GFCN model takes as input the adjacency matrix $\mathbf{A}$ and initial feature matrix $\mathbf{X} = \mathbf{H}^{(0)}$. At each layer, a node aggregates information from its neighboring nodes and the initial feature matrix through skip connection. The aggregated information is then transformed using learnable weight matrices. The resulting representation is then passed to the next layer for further propagation. Finally, the output is the latent graph representation $\mathbf{H}^{(L)}$ from the last network layer.

most of the existing graph convolutions with skip connections [22, 113] are based on heuristics, our graph fair convolution is theoretically motivated by implicit fairing and derived directly from the Jacobi iterative method. It is important to mention that the convolution operation in the proposed propagation rule involves aggregating information from a node's neighbors and combining it with the node's own features to produce a new set of features. As a result, the normalized adjacency matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is particularly useful in this context because it provides a way to normalize the aggregation of neighbor features, which helps avoid the problem of over-reliance on highly connected nodes. This normalization also helps make the convolution operation more stable and better conditioned, which can improve the convergence and generalization performance of our mode. The normalized adjacency matrix considers both the number of neighbors connected to a node and the number of neighbors connected to each of those neighboring nodes.

In contrast to GCN-based methods which use a first-order approximation of spectral graph convolutions and a renormalization trick in their layer-wise propagation rules to avoid numerical instability, our GFCN model does not require any renormalization as the spectral radius of the normalized adjacency matrix is equal to 1, while still maintaining the key property of convolution as a neighborhood aggregation operator. Hence, repeated application of the GFCN's layer-wise propagation rule provides a computationally efficient convolutional process, leading to numerical stability and avoidance of exploding/vanishing gradients. It should also be pointed out that similar to GCN, the proposed GFCN model also does not require explicit computation of the Laplacian eigenbasis. In addition, the aggregation of GFCN with skip connections between the initial fea-

ture matrix and each hidden layer does not require filtered learning of the hidden layers. Skip connections are a mechanism that allows our model to directly pass on the information from the initial feature matrix (input) to the deeper hidden layers, thereby helping to retain the original node features to prevent loss of important information during the aggregation process.

3.4.6 Model Prediction

The layer-wise propagation rule of GFCN is basically a node embedding transformation that projects both the input $\mathbf{H}^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$ into a trainable weight matrix $\mathbf{\Theta}^{(\ell)} \in \mathbb{R}^{F_\ell \times F_{\ell+1}}$ and the initial feature matrix $\mathbf{X}$ into the skip-connection weight matrix $\widetilde{\mathbf{\Theta}}^{(\ell)} \in \mathbb{R}^{F \times F_{\ell+1}}$, with $F_{\ell+1}$ feature maps. Then, a point-wise activation function $\sigma(\cdot)$ such as $\text{ReLU}(\cdot) = \max(0, \cdot)$ is applied to obtain the output node embedding matrix. The aggregation scheme of GFCN is depicted in Figure 3.3. Note that GFCN uses skip connections between the initial feature matrix and each hidden layer. Skip connections not only allow the model to carry over information from the initial node attributes, but also help facilitate training of multi-layer networks.

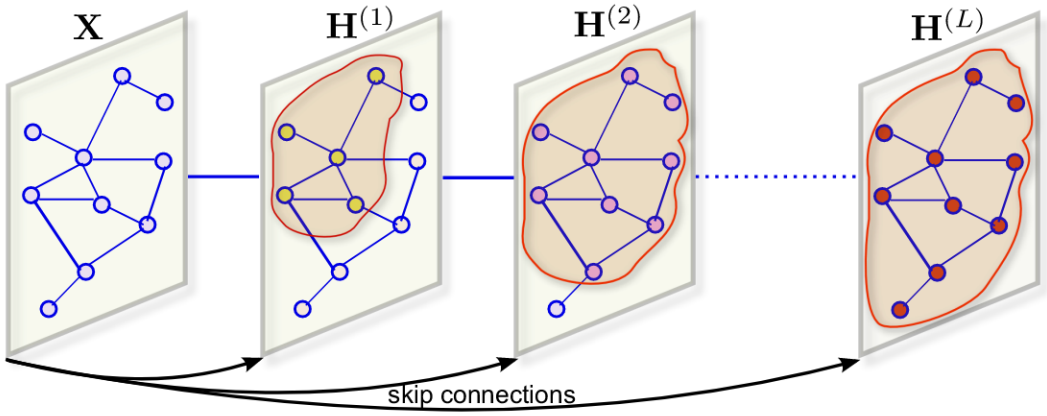


Figure 3.3: Illustration of the GFCN aggregation scheme with skip connections.

The embedding $\mathbf{H}^{(L)}$ of the last layer of GFCN contains the final output node embeddings, which can be used as input for downstream tasks such as graph classification, clustering, visualization, link prediction, recommendation, node classification, and anomaly detection. Since anomaly detection can be posed as a binary classification problem, we apply the point-wise softmax function (i.e. sigmoid in the binary case) to obtain an $N \times 2$ matrix of predicted labels for graph nodes

$$\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_N)^\intercal = \text{softmax}(\mathbf{H}^{(L)}), \tag{3.10}$$

where $\hat{\mathbf{y}}_i = (\hat{y}_i, 1 - \hat{y}_i)$ is a two-dimensional raw vector of predicted probabilities, with $\hat{y}_i$ the probability that the network associates the i-th node with one of the two classes (i.e. 1 for anomalous and 0 for normal).

3.4.7 Model Complexity

For simplicity, we assume the feature dimensions are the same for all layers, i.e. $F_\ell = F$ for all ℓ, with $F \ll N$. The time complexity of an L-layer GFCN is $\mathcal{O}(L\|\mathbf{A}\|_0 F + LNF^2)$, where $\|\mathbf{A}\|_0$ denotes the number of non-zero entries of the sparse adjacency matrix. Note that multiplying the normalized adjacency matrix with an embedding costs $\mathcal{O}(\|\mathbf{A}\|_0 F)$ in time, while multiplying an embedding with a weight matrix costs $\mathcal{O}(NF^2)$. Also, multiplying the initial feature matrix by the skip-connection weight matrix costs $\mathcal{O}(NF^2)$.

For memory complexity, an L-layer GFCN requires $\mathcal{O}(LNF + LF^2)$ in memory, where $\mathcal{O}(LNF)$ is for storing all embeddings and $\mathcal{O}(LF^2)$ is for storing all layer-wise weight matrices.

Therefore, our proposed GFCN model has the same time and memory complexity as GCN, albeit GFCN takes into account distant graph nodes for improved learned node representations.

3.4.8 Model Training

The parameters (i.e. weight matrices for different layers) of the proposed GFCN model for semi-supervised anomaly detection are learned by minimizing the following regularized loss function

$$\mathcal{L} = \frac{1}{N_l} \sum_{i=1}^{N_l} \mathcal{C}_\alpha(\mathbf{y}_i, \hat{\mathbf{y}}_i) + \frac{\beta}{2} \sum_{\ell=0}^{L-1} \left(\|\mathbf{\Theta}^{(\ell)}\|_F^2 + \|\widetilde{\mathbf{\Theta}}^{(\ell)}\|_F^2 \right), \tag{3.11}$$

where $\mathcal{C}_\alpha(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ is the weighted cross-entropy given by

$$\mathcal{C}_\alpha(\mathbf{y}_i, \hat{\mathbf{y}}_i) = -\alpha\, y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i), \tag{3.12}$$

which measures the dissimilarity between the one-hot encoding vector $\mathbf{y}_i$ of the ith node and the corresponding vector $\hat{\mathbf{y}}_i$ of predicted probabilities. This dissimilarity decreases as the value of the predicted probability approaches the ground-truth label. The weight parameter α adjusts the importance of the positive labels by assigning more weight to the anomalous class, while the parameter β controls the importance of the regularization term, which is added to prevent overfitting. The regularization term is the sum of the squared elements of the learnable weight matrices for each layer. It is important to mention that we only use the normal class labeled instances to train our model.

We optimize our model using the Adam optimizer [115], which is a modified version of Stochastic Gradient Descent (SGD) that uses adaptive moment estimation. The intuition behind the use of the weighted cross-entropy loss function is to assign a higher weight to the anomalous nodes than to the normal nodes, so that the model is encouraged to correctly identify the anomalous nodes

even if they are rare and overshadowed by the large number of normal nodes. The regularization term, on the other hand, penalizes large weight values in the learnable weight matrices, which helps to reduce the complexity of the model and improve its generalization performance. The strength of the regularization is controlled by the value of the hyperparameter β, which is tuned using grid search.

The weight matrices of our GFCN model are initialized randomly with small values using a normal distribution to ensure that the variance of the activations and gradients is roughly the same across all layers of the network. During training, the optimizer adjusts the weight matrices to minimize the regularized loss function. The training process involves choosing the hyperparameters, computing the regularized weighted cross-entropy loss, feeding forward and backpropagating the inputs, and updating the weight matrices using the Adam optimizer. This process is repeated for multiple epochs until the model converges or the validation loss does not decrease after a specified number of consecutive epochs.

3.4.9 Model Inference

Once the model is trained, we can use the weighted cross-entropy errors to assess the abnormality of nodes. To this end, we define the anomaly score of the ith test node as

$$s_i = \mathcal{C}_\alpha(\mathbf{y}_i, \hat{\mathbf{y}}_i). \tag{3.13}$$

Since the range of the weighted cross-entropy is $[0, \infty]$ (e.g. infinite value when $y_i = 1$ and $\hat{y}_i = 0$), we apply min-max normalization to bring all anomaly scores into the range [0,1] as follows:

$$\tilde{s}_i = \frac{s_i - s_{\min}}{s_{\max} - s_{\min}}, \tag{3.14}$$

where $s_{\min}$ and $s_{\max}$ are the minimum and maximum, respectively, of the anomaly scores in the test set. Nodes with scores larger than a certain threshold are considered anomalies. Hence, we can compute a ranked list of anomalies according to their normalized anomaly scores. In other words, we compute the anomaly scores of each node in the test set, and then the top-r nodes with higher scores are identified as anomalies for a user-specified value of r.

3.5 Experiments

In this section, we conduct extensive experiments to assess the performance of the proposed anomaly detection framework in comparison with state-of-the-art methods on several benchmark

datasets. The source code to reproduce the experimental results is made publicly available on GitHub[1].

3.5.1 Datasets

We demonstrate and analyze the performance of the proposed model on three citation networks: Cora, Citeseer, and Pubmed [116], and two co-purchase graphs: Amazon Photo and Amazon Computers [117]. The summary descriptions of these benchmark datasets are as follows:

- Cora is a citation network dataset consisting of 2708 nodes representing scientific publications and 5429 edges representing citation links between publications. All publications are classified into 7 classes (research topics). Each node is described by a binary feature vector indicating the absence/presence of the corresponding word from the dictionary, which consists of 1433 unique words.

- Citeseer is a citation network dataset composed of 3312 nodes representing scientific publications and 4723 edges representing citation links between publications. All publications are classified into 6 classes (research topics). Each node is described by a binary feature vector indicating the absence/presence of the corresponding word from the dictionary, which consists of 3703 unique words.

- Pubmed is a citation network dataset containing 19717 scientific publications pertaining to diabetes and 44338 edges representing citation links between publications. All publications are classified into 3 classes. Each node is described by a TF/IDF weighted word vector from the dictionary, which consists of 500 unique words.

- Amazon Computers and Amazon Photo datasets are co-purchase graphs [117], where nodes represent goods and edges indicate that two goods are frequently bought together. The node features are bag-of-words encoded product reviews, while the class labels are given by the product category.

- ogbn-arxiv dataset is a large-scale graph dataset from open graph benchmark (OGB) representing the citation network between all computer science (CS) arXiv papers. In this dataset over 169k nodes and 1.1m edges are contained. Each node has a 128-dimensional feature vector obtained by averaging the embeddings of words in the article's title and abstract.

[1] https://github.com/MahsaMesgaran/GFCN

Since there is no ground truth of anomalies in these datasets, we employ the commonly-used protocol [29] in anomaly detection by treating the smallest class for each dataset as the anomaly class and the remaining classes as the normal class. Dataset statistics are summarized in Table 3.1, where anomaly rate refers to the percentage of abnormalities in each dataset. For all datasets, we only use the normal class labeled instances to train the model.

Table 3.1: Summary statistics of datasets.

Dataset	Nodes	Edges	Features	Classes	Anomaly Rate (%)
Cora	2708	5278	1433	7	0.06
Citeseer	3327	4732	3703	6	0.07
Pubmed	19717	44338	500	3	0.21
Photo	7487	119043	745	8	0.04
Computers	13381	245778	767	10	0.02
ogbn-arxiv	169343	1166243	128	40	0.02

3.5.2 Baseline Methods

We evaluate the performance of the proposed method against various baselines, including one-class support vector machines (OC-SVMs) [98], imbalanced vertex diminished (ImVerde) [118], one-class deep support vector data description (Deep SVDD) [26], deep anomaly detection on attributed networks (Dominant) [28], one-class deep semi-supervised anomaly detection (Deep SAD) [27], graph convolutional networks (GCNs) [1], GCN-based anomaly detection (GCN-N and GCN-AN) [29], graph random neural networks (GRAND) [119], semi-Supervised node classification on graph with few labels via non-parametric distribution assignment (GraFN) [120], and re-weighting the influence of labeled nodes (ReNode) [121]. For baselines, we mainly consider methods that are closely related to GFCN and/or the ones that are state-of-the-art anomaly detection frameworks. A brief description of these strong baselines can be summarized as follows:

- **OC-SVM** [98] is an unsupervised one-class anomaly detection technique, which learns a discriminative hyperplane boundary around the normal instances using support vector machines by maximizing the distance from this hyperplane to the origin of the high-dimensional feature space.

- **ImVerde** [118] is a semi-supervised graph representation learning technique for imbalanced graph data based on a variant of random walks by adjusting the transition probability each time a graph node is visited by the random particle.

- **Deep SVDD** [26] is an unsupervised anomaly detection method, inspired by kernel-based one-class classification and minimum volume estimation, which learns a spherical, instead of a hyperplane, boundary in the feature space around the data using support vector data description. It trains a deep neural network while minimizing the volume of a hypersphere that encloses the network embeddings of the data. Normal instances fall inside the hypersphere, while anomalies fall outside.

- **Dominant** [28] is a deep autoencoder based on GCNs for unsupervised anomaly detection on attributed graphs. It employs an objective function defined as a convex combination of the reconstruction errors of both graph structure and node attributes. These learned reconstruction errors are then used to assess the abnormality of graph nodes.

- **Deep SAD** [27] is a semi-supervised anomaly detection technique, which generalizes the unsupervised Deep SVDD approach to the semi-supervised setting by incorporating a new term for labeled training data into the objective function. The weights of the Deep SAD network are initialized using an autoencoder pre-training mechanism.

- **GCN** [1] is a deep graph neural network for semi-supervised learning of graph representations, encoding both local graph structure and attributes of nodes. It is an efficient extension of convolutional neural networks to graph-structured data, and uses a graph convolution that aggregates and transforms the feature vectors from the local neighborhood of a graph node.

- **GCN-N** and **GCN-AN** [29] are GCN-based, semi-supervised anomaly detection frameworks, which rely on minimizing the volume of a hypersphere that encloses the node embeddings to detect anomalies. Node embeddings placed inside and outside this hypersphere are deemed normal and anomalous, respectively. GCN-N uses only normal label information, while GCN-AN uses both anomalous and normal label information.

- **GRAND** [119] is a semi-supervised learning on graphs when labeled nodes are scarce. This technique relies on a random propagation strategy to perform graph data augmentation and employs consistency regularization to optimize prediction consistency of unlabeled nodes across different data augmentations.

- **GraFN** [120] is a semi-supervised node representation learning for graphs with few labeled nodes. This technique exploits the self-supervised loss to ensure nodes that belong to the same class to be grouped together on differently augmented graphs. GraFN randomly samples support nodes from the labeled nodes and anchor nodes from the entire graph, and

non-parametrically compute two predicted class distributions from two augmented graphs based on the anchor supports similarity.

- **ReNode** [121] is a semi-supervised node classification technique addressing the the topology-imbalance node representation learning as a graph specific imbalance learning problem. To measure the degree of topology imbalance, a conflict detection-based metric, Totoro, is used to locate node position. ReNode adjusts the training weights of labeled nodes based on their topological positions.

3.5.3 Evaluation Metric

In order to evaluate the performance of our proposed framework against the baseline methods, we use AUC, the area under the receiving operating characteristic (ROC) curve, as a metric. AUC summarizes the information contained in the ROC curve, which plots the true positive rate versus the false positive rate, at various thresholds. Larger AUC values indicate better performance at distinguishing between anomalous and normal instances. An uninformative anomaly detector has an AUC equal to 50% or less. An AUC of 50% corresponds to a random detector (i.e. for every correct prediction, the next prediction will be incorrect), whereas an AUC score smaller than 50% indicates that a detector performs worse than a random detector.

3.5.4 Implementation Details

For fair comparison, we implement the proposed method and baselines in PyTorch using the PyTorch Geometric library. Following common practices for evaluating performance of GCN-based models [1, 29], we train our 2-layer GFCN model for 100 epochs using the Adam optimizer [115] with a learning rate of 0.1. We tune the latent representation dimension by hand, and set it to 128. The hyperparameters α and β are chosen via grid search with cross-validation over the sets $\{10^{-4}, 10^{-3}, \ldots, 1\}$ and $\{2, 3, \ldots, 10\}$, respectively. We tune hyperparameters using the validation set, and terminate training if validation loss does not decrease after 10 consecutive epochs. For each dataset, we consider the settings where 2.5%, 5% and 10% of instances are labeled, and we compute the average and standard deviation of test AUCs over ten runs.

3.5.5 Anomaly Detection Performance

Tables 3.2-3.4 present the anomaly detection results on the five datasets. The best results are highlighted as bold. For each dataset, we report the AUC averaged over 10 runs as well as the standard deviation, at various ratios of labeled instances. As can be seen, our GFCN model consistently

achieves the best performance on all datasets, except in the case of the Cora dataset when 10% of instances are labeled. In that case, GCN-AN yields a marginal improvement of 0.7% over GFCN, despite the fact that GCN-AN is trained on both normal and anomalous instances, whereas our model is trained only on normal instances. In addition, ImVerde, Deep SAD, GCN and GCN-AN all perform reasonably well on all datasets at various levels of label rates, but we find that GFCN outperforms these baselines on almost all datasets, while being considerably simpler. An AUC score of 50% or less indicates that the baseline is an uninformative anomaly detector.

On the Amazon Computers dataset, Table 3.2 shows that the proposed GFCN approach performs on par with GCN-AN, but outperforms all baselines on the other four datasets. In particular, GFCN yields 16.2% and 15.7% performance gains over Deep SAD on the Cora and Photo datasets, respectively. These gains are consistent with the results shown in Tables 3.3-3.4. We argue that the better performance of GFCN over GCN-N and Deep SAD is largely attributed to the fact that our model does not suffer from the hypersphere collapse problem. Interestingly, the performance gains are particularly higher at the lower label rate 2.5%, confirming the usefulness of semi-supervised learning in that it improves model performance by leveraging unlabeled data. Another interesting observation is that in general both GCN-AN and GCN-N yield relatively high AUC standard deviations compared to our GFCN model, indicating that our model has less variability than these two strong baselines.

Lastly, we examined the training times (in seconds) for GFCN on Cora when 10% of all instances were labeled. We also recorded the training times of GFCN, GCN-AN, GCN-N, and GCN on Cora for which we obtained 3.19, 4.12, 2.31, and 2.06 seconds, respectively. Since GFCN uses the skip connection, it took more training time than GCNs. However, the experiment shows the proposed method could learn the abnormalities fast enough.

3.5.6 Parameter Sensitivity Analysis

The weight hyperparameter α of the weighted cross-entropy and the regularization hyperparameter β play an important role in the anomaly detection performance of the proposed GFCN framework. We conduct a sensitivity analysis to investigate how the performance of GFCN changes as we vary these two hyperparameters. In Figure 3.4, we analyze the effect of the hyperparameter α by plotting the AUC results of GFCN vs. α using various label rates for all datasets, where α varies from 2 to 10. We can see that with a few exceptions, our model generally benefits from relatively larger values of the weight hyperparameter. For almost all datasets, our model achieves satisfactory performance with $\alpha = 4$.

In Figure 3.5, we plot the average AUCs, along with the standard error bars, of our GFCN model

Table 3.2: Test AUC (%) averaged over 10 runs when 2.5% of instances are labeled. We also report the standard deviation. Boldface numbers indicate the best anomaly detection performance.

	Dataset					
Method	Cora	Citeseer	Pubmed	Photo	Computers	ogbn-arxiv
OC-SVM [98]	50.0±0.1	50.6±0.4	68.9±0.9	51.9±0.6	47.3±0.7	-
ImVerde [118]	85.9±6.1	60.3±6.5	94.3±0.5	89.1±1.4	98.5±0.7	-
Deep SVDD [26]	69.6±6.5	55.3±1.6	73.7±6.3	52.3±1.4	46.6±1.5	-
Dominant [28]	52.3±0.9	53.9±0.6	50.8±0.4	38.1±0.4	46.8±1.2	-
Deep SAD [27]	72.7±6.0	53.8±2.9	91.3±2.4	81.9±5.7	92.2±2.5	-
GCN [1]	84.9±6.9	60.9±6.0	96.2±0.1	90.1±2.5	98.1±0.3	51.2±0.1
GCN-AN [29]	88.8±5.4	65.6±4.7	95.6±0.3	95.4±1.8	**98.8**±0.3	-
GCN-N [29]	62.6±9.9	56.0±4.4	76.5±4.2	55.1±11	56.9±5.1	-
GRAND [119]	81.3±8.1	56.4±4.5	89.6±1.2	88.7±2.1	91.7±5.5	50.7±1.3
GraFN [120]	58.2±5.8	56.3±3.2	81.4±0.9	97.4±1.5	96.1±3.9	57.3±1.4
ReNode [121]	68.4±6.8	55.1±2.0	82.3±1.5	84.9±3.4	96.3±9.2	50.2±3.1
GFCN	**93.9**±2.3	**68.3**±1.1	**96.3**±0.1	**97.6**±0.5	**98.8**±0.4	**60.3**±0.1

Table 3.3: Test AUC (%) averaged over 10 runs when 5% of instances are labeled. We also report the standard deviation. Boldface numbers indicate the best anomaly detection performance.

	Dataset					
Method	Cora	Citeseer	Pubmed	Photo	Computers	ogbn-arxiv
OC-SVM [98]	50.2±0.1	50.7±0.5	71.0±1.1	51.9±0.6	47.2±0.8	-
ImVerde [118]	91.1±3.2	64.5±4.5	94.9±0.5	92.2±1.2	98.6±0.6	-
Deep SVDD [26]	58.3±2.8	56.0±0.8	79.9±3.4	52.3±0.8	47.0±1.4	-
Dominant [28]	52.5±0.9	53.9±0.7	53.0±0.5	38.1±0.5	47.2±1.2	-
Deep SAD [27]	72.4±5.7	61.1±4.2	91.7±1.7	89.8±3.1	92.8±2.6	-
GCN [1]	89.2±7.6	63.9±4.7	96.6±0.1	92.3±1.2	98.3±0.3	53.0±0.1
GCN-AN [29]	91.8±5.4	68.3±3.8	96.2±0.2	97.0±0.7	99.1±0.3	-
GCN-N [29]	67.1±5.8	57.4±3.1	76.1±4.8	56.2±9.6	58.2±5.8	-
GRAND [119]	84.7±9.1	56.8±3.1	87.8±1.4	98.4±4.1	97.8±5.8	60.2±5.3
GraFN [120]	61.3±12.6	56.1±4.3	83.0±6.1	98.2±5.6	96.6±7.9	59.8±6.1
ReNode [121]	69.9±3.3	56.1±1.6	82.7±1.1	85.7±2.1	97.5±2.1	60.1±1.9
GFCN	**92.2**±2.3	**71.3**±1.3	**96.7**±0.1	**98.8**±0.4	**99.3**±0.4	**61.1**±0.4

vs. β using various label rates for all datasets, and by varying the value of β from 10^{-4} to 1. Notice that the best performance is generally achieved when $\beta = 0.01$, except in the cases of the Citeseer and Pubmed datasets, on which the best performance is obtained when $\beta = 0.1$. In general, when the regularization hyperparameter increases, the performance improves rapidly at the very beginning, but then deteriorates after reaching the best setting due to overfitting. An interesting

Table 3.4: Test AUC (%) averaged over 10 runs when 10% of instances are labeled. We also report the standard deviation. Boldface numbers indicate the best anomaly detection performance.

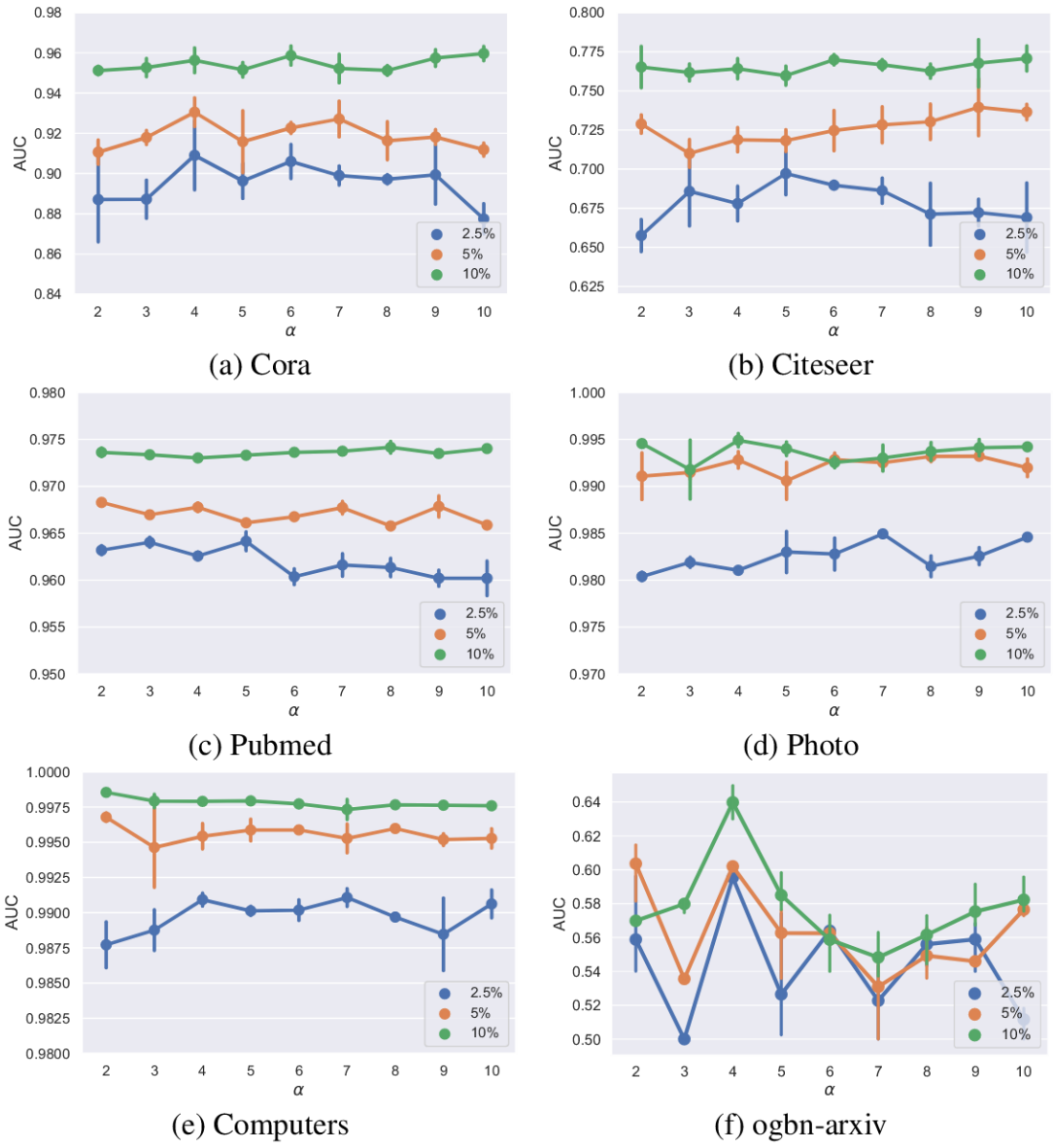
	Dataset					
Method	Cora	Citeseer	Pubmed	Photo	Computers	ogbn-arxiv
OC-SVM [98]	51.8±1.7	51.1±0.9	73.1±0.8	51.7±0.9	47.4±0.8	-
ImVerde [118]	94.5 ±2.1	68.5 ±6.9	95.5 ±0.4	92.8 ±1.0	99.1 ±0.4	-
Deep SVDD [26]	59.8 ±4.8	56.7 ±1.7	93.2 ±1.0	53.1 ±0.6	47.5 ±1.0	-
Dominant [28]	52.6 ±0.9	53.9 ±0.8	53.1 ±0.4	38.1 ±0.7	46.6 ±1.9	-
Deep SAD [27]	72.9 ±3.3	62.4 ±4.4	96.7 ±0.1	89.5 ±1.9	93.5 ±1.9	-
GCN [1]	94.5 ±4.3	68.6 ±3.3	96.7 ±0.1	93.3 ±0.8	98.3 ±0.3	53.9±0.3
GCN-AN [29]	**95.4** ±2.7	72.9 ±4.3	96.6 ±0.1	97.9 ±0.3	99.1 ±0.3	-
GCN-N [29]	72.3 ±7.0	60.1 ±2.1	73.4 ±5.7	53.6 ±3.9	58.5 ±4.6	-
GRAND [119]	86.5 ±1.3	57.8±6.0	88.8±1.1	94.6±0.7	93.1±0.2	59.9±0.5
GraFN [120]	78.7±7.4	60.3±8.1	83.1±6.3	98.7±4.0	97.2±1.7	62.6±3.3
ReNode [121]	68.3±4.2	54.6±0.8	83.4±1.4	90.4±2.3	99.0±0.3	61.2±1.8
GFCN	94.7 ±1.0	**76.5** ±1.0	**97.3** ±0.1	**99.4** ±0.2	**99.4** ±0.4	**63.0**±0.8

observation is that GFCN generally shows steady increase in performance with the regularization parameter, except on the Citeseer dataset when the label rate is 5%, whereas the performance on the other datasets degrades after reaching a certain threshold.

We also analyzed the effects of the number of layers and latent representation dimension on the performance of our model using the Cora, Citeseer, and Pubmed datasets (10% of the instances as labeled), and the average AUC results are displayed in Figure 3.6. As shown in Figure 3.6 (left), the performance of GFCN remains relatively stable as we increase the depth of the network from 2 to 10 layers. Figure 3.6 (right) shows the AUC results with the latent representation dimension varying from 10 to 256. As can be seen, our model typically benefits from larger latent dimensions, achieving a good performance with a latent representation dimension equal to 128 for all datasets.

3.5.7 Visualization

The feature embeddings learned by GFCN can be visualized using the Uniform Manifold Approximation and Projection (UMAP) algorithm [122], which is a dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a two- or three-dimensional space. Figure 3.7 displays the UMAP embeddings of GFCN (left) and GCN (right) using the Cora dataset. As can be seen, the GFCN embeddings are more separable than the GCN ones. With GCN features, the normal and anomalous instances are not discriminated very well, while with GFCN features these data instances are discriminated much better, indicating that



(a) Cora (b) Citeseer

(c) Pubmed (d) Photo

(e) Computers (f) ogbn-arxiv

Figure 3.4: Effect of weight hyperparameter α on anomaly detection performance (AUC).

GFCN learns better node representations for anomaly detection tasks.

3.5.8 Ablation Studies

To validate the influence of our proposed components, we conduct additional experiments for ablation studies by removing components individually. The details and results of our experiments are shown in Table 4.6, where we report both the average and standard deviation of AUCs over 10 runs on the Cora, Citeseer, and Pubmed datasets. As can be seen, the removal of each component leads to a deterioration in performance. By adding skip connections, an improvement in perfor-

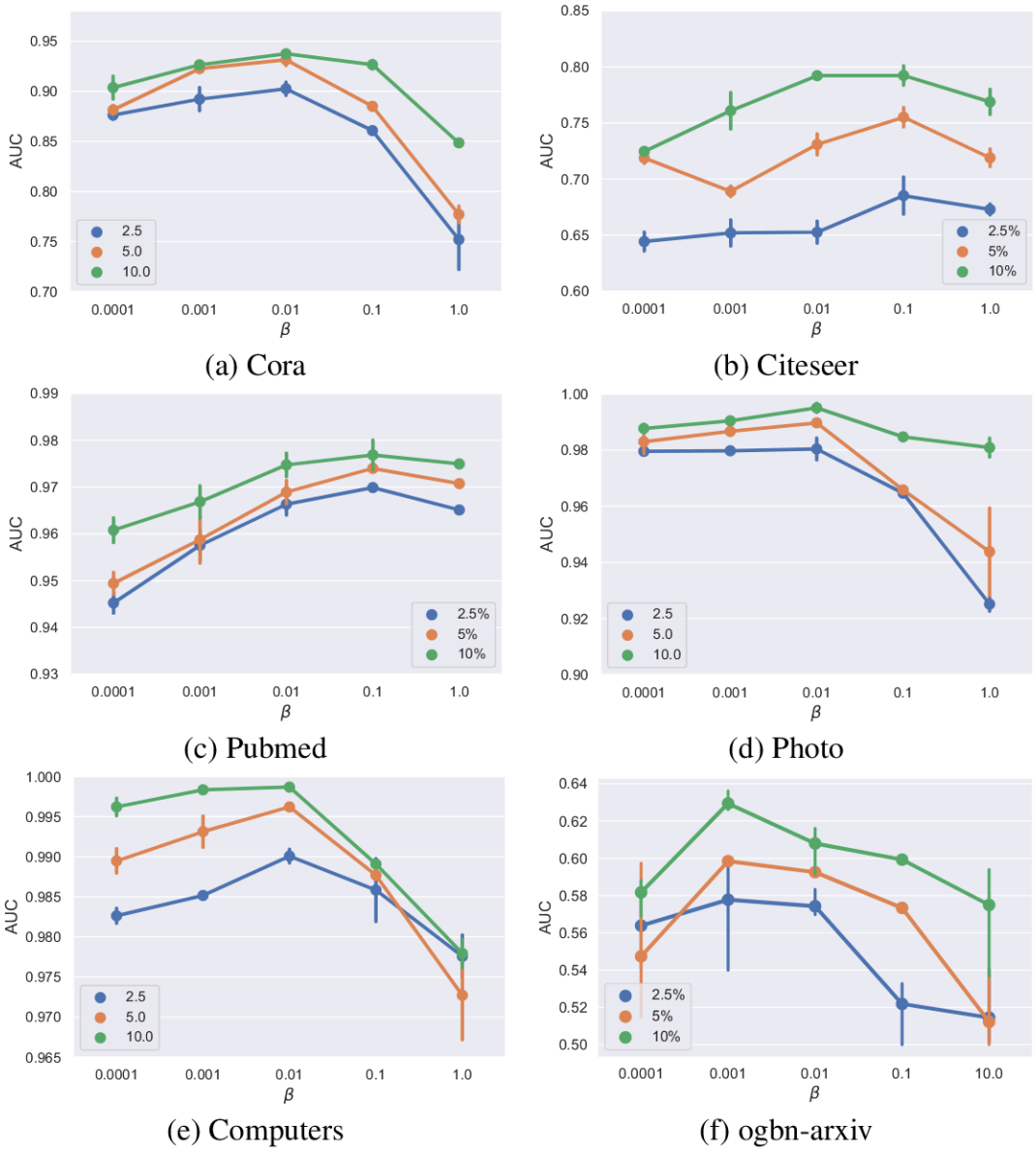


Figure 3.5: Effect of regularization hyperparameter β on anomaly detection performance (AUC).

mance can be observed on all datasets. This indicates that reusing the initial node features in each layer and memorizing information across distant nodes yield improved results. Consistent with prior work, skip connections have been shown to mitigate the vanishing gradient problem in deep neural networks, as well as to improve the accuracy and convergence speed of the network. This has been demonstrated in a number of previous studies, including the original convolutional neural network with deep residual learning (ResNet) [123] and subsequent work on graph representation learning such as jumping knowledge network (JK-Net) [22], residual graph convolutional network (ResGCN) [31], and graph convolutional networks with initial residual and identity mapping (GC-

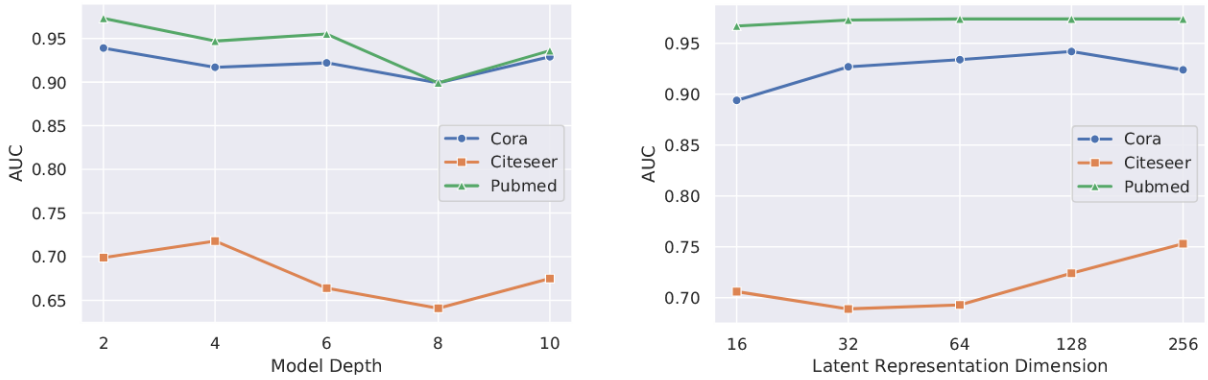


Figure 3.6: Effects of number of layers (left) and latent representation dimension (right) on anomaly detection performance of our GFCN model using the Cora, Citeseer, Pubmed dataset when 10% of instances are labeled. The AUC results are averaged over 10 runs.

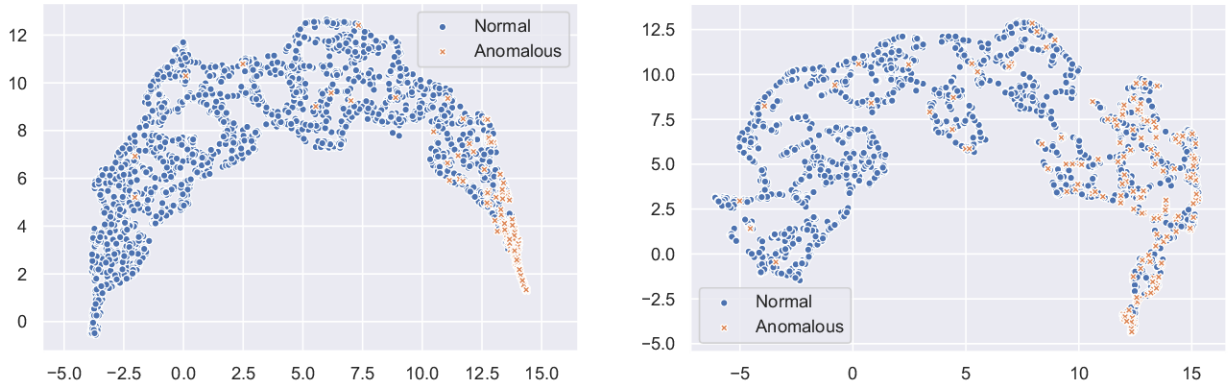


Figure 3.7: UMAP embeddings of GFCN (left) and GCN (right) using the Cora dataset.

NII) [32]. Also, Xu *et al.* [124] examined the optimization dynamics of graph neural networks (GNNs) during their training process, and showed theoretically that skip connections implicitly accelerate the training of GNNs.

As reported in Table 4.6, the importance of the regularization term becomes apparent when removing it from the proposed loss function, resulting in a significant drop in AUC. Regularization is important to avoid model overfitting by imposing a penalty to learnable weights. Using regularization term in our loss function yields 9.2%, 8.7%, and 2.9% performance gains over its counterpart model without regularization on the Cora, Citeseer, and Pubmed datasets, respectively.

On the other hand, it is worth pointing out that removing the skip connection and regularization components yields relatively high AUC standard deviations compared to the proposed GFCN model, indicating the robustness of our model.

Table 3.5: Test AUC (%) averaged over 10 runs when 5% of instances are labeled. We also report the standard deviation.

Method	Cora	Citeseer	Pubmed
Without skip connection	91.2±2.7	64.6±3.8	94.2±0.2
Without regularization	84.7±6.2	59.6±4.1	93.4±0.1
GFCN	**93.9**±2.3	**68.3**±1.1	**96.3**±0.1

CHAPTER 4

A Graph Encoder-Decoder Network for Unsupervised Anomaly Detection

A key component of many graph neural networks (GNNs) is the pooling operation, which seeks to reduce the size of a graph while preserving important structural information. However, most existing graph pooling strategies rely on an assignment matrix obtained by employing a GNN layer, which is characterized by trainable parameters, often leading to significant computational complexity and a lack of interpretability in the pooling process. In this chapter, we propose an unsupervised graph encoder-decoder model to detect abnormal nodes from graphs by learning an anomaly scoring function to rank nodes based on their degree of abnormality. In the encoding stage, we design a novel pooling mechanism, named LCPool, which leverages locality-constrained linear coding for feature encoding to find a cluster assignment matrix by solving a least-squares optimization problem with a locality regularization term. By enforcing locality constraints during the coding process, LCPool is designed to be free from learnable parameters, capable of efficiently handling large graphs, and can effectively generate a coarser graph representation while retaining the most significant structural characteristics of the graph. In the decoding stage, we propose an unpooling operation, called LCUnpool, to reconstruct both the structure and nodal features of the original graph. We conduct empirical evaluations of our method on six benchmark datasets using several evaluation metrics, and the results demonstrate its superiority over state-of-the-art anomaly detection approaches.

4.1 Introduction

Graph anomaly detection generally refers to the task of identifying graph nodes that exhibit unusual or unexpected behavior based on their structure and/or feature information. Capturing these abnormal nodes is challenging, primarily because anomalies are rare occurrences and only a very small proportion of the graph nodes might be anomalous. Specific applications of graph anomaly detection include fraud detection, network intrusions, abnormal behavior in social networks, biological systems, communication networks, and financial transactions [125, 126]. For example, in social networks, graph anomaly detection can be used to identify fraudulent accounts or suspicious activities.

Detecting anomalies in a graph typically involves identifying nodes that deviate significantly from the normal behavior of the graph, either in terms of their structural characteristics and/or their feature attributes [127]. However, graphs can often be very large and complex, making it challenging to identify such anomalies. To address this problem, graph pooling can be used to reduce the size of the graph while preserving its important structural features [41–43]. The aim is to produce a coarse representation of the graph structure by summarizing the information contained in the nodes of the graph into a fixed-size vector or matrix while preserving the salient features of the graph. By producing a coarse representation of the graph structure, graph pooling can help abstract away irrelevant or noisy details, and focus on the most important structural properties of the graph. Graph pooling methods can be categorized into two main types: global and hierarchical pooling. The former aggregates all of the node features in the graph into a single vector or scalar [38–40]. This is typically done using summary statistics (e.g., mean or maximum) to aggregate the features of a set of graph nodes. The resulting global feature can then be used as input to a downstream classifier or regressor. Hierarchical pooling, on the other hand, operates at multiple levels of the graph hierarchy [41–43]. The idea is to recursively partition the graph into smaller subgraphs, and then apply pooling at each level to obtain a hierarchy of graph representations. This can be done using clustering-based techniques to group nodes with similar features and represent each group with a single node, resulting in a smaller graph with fewer nodes. More recently, unsupervised methods, which do not require labeled data during training, have been shown effective at addressing anomaly and fault detection problems in various data settings. Tao *et al.* [128] introduce an unsupervised cross-domain diagnosis method to learn fault features specific to the target domain using unlabeled data from the source domain. Song *et al.* [129] present an adaptive neural finite-time resilient dynamic surface control strategy to overcome unknown control coefficients induced by severe faults and false data injection attacks. Also, Song *et al.* [130] propose an adaptive fixed-time prescribed performance trajectory tracking controller, incorporating an event-triggered control mechanism,

while considering the trade-off between tracking performance and communication cost.

There are several approaches to graph anomaly detection that can be used to identify anomalous nodes in a graph. One common approach is to use unsupervised methods [28, 131–133], which involve identifying anomalous nodes in a graph without using any labeled data. This can be particularly challenging, as there is no ground truth for what constitutes an anomaly. Recently, graph neural networks (GNNs) have achieved state-of-the-art performance in anomaly detection tasks, due largely to their ability to learn efficient representations of nodes that capture their structural or attribute similarity [127, 134]. GNNs learn node representations by aggregating information from the neighboring nodes of each node in the graph, and hence they are able to capture both the attributes of each node as well as the structure of the graph as a whole. By analyzing both the node attributes and the graph structure simultaneously, GNNs can identify nodes that are significantly different from their neighbors or that exhibit unusual patterns of behavior.

While strong in learning node representations, GNN-based methods are, however, unable to aggregate the information in a hierarchical way [41]. To this end, an effective aggregation function is required to aggregate the information at the node level as well as the entire graph. Such an aggregation strategy is performed using a graph pooling operation, which involves merging nodes or clusters of nodes in the input graph to create a coarser, smaller graph while preserving important structural information of the input graph. Graph pooling can be thought of as a type of downsampling, where the goal is to create a smaller version of the input graph that captures the most salient features of the original graph. Clustering algorithms such as K-means or spectral clustering are usually employed to group nodes based on their structural similarities. Hierarchical pooling methods, for instance, reduce the graph size by dropping graph nodes based on a learnable score or by coarsening the graph using a cluster assignment matrix to group nodes into a pre-defined number of clusters. DiffPool [41] is a graph pooling method, which uses differential graph convolutions to learn an assignment matrix mapping each node to a set of clusters that are used to create a coarser, smaller graph. However, this assignment matrix is dense, and hence it is not easily scalable to large graphs [135]. SAGPool [42] is a graph pooling method, which leverages self-attention networks to learn node scores and construct a smaller sized graph by selecting a subset of nodes based on their scores. However, this technique is unable to preserve node and edge information effectively and they suffer from information loss. Moreover, SAGPool may not be well-suited for handling large graphs with many nodes and edges, as the self-attention mechanism can become computationally expensive for such graphs.

Inspired by locality-constrained linear coding (LLC) [136], we introduce an unsupervised graph encoder-decoder model for anomaly detection. LLC is a variant of sparse coding that imposes

a locality constraint on the weight coefficients. This locality constraint helps capture the local structure of the input signals, leading to a better generalization performance compared to traditional sparse coding. Our proposed model architecture is comprised of two main components: an encoder and a decoder. In the encoding stage, a graph convolutional network encoder is used to generate a latent representation, followed by a graph pooling layer to coarsen the graph. In the decoding stage, an unpooling layer is applied to the coarser graph, followed by a graph deconvolutional network decoder to reconstruct the graph. Our objective is to design a graph pooling operation that is trainable, devoid of learnable parameters, and capable of scaling up to handle large graphs. To this end, we propose a locality-constrained pooling strategy, dubbed LCPool, which generates a coarser graph using a cluster assignment matrix obtained via LLC by solving a constrained least square fitting problem with a locality regularization term. In contrast to sparse coding models, which often rely on computationally intensive optimization algorithms, the objective function used by LLC has an analytical solution [136], making it perform very fast in practice. Since the deconvolution operation in the decoding stage may introduce undesirable noise into the output graph due to overlapping receptive fields and information loss during downsampling, we employ spectral graph wavelets to enable the preservation of essential details and the removal of unwanted noise, resulting in improved reconstruction of the node feature matrix. The basis functions for the spectral graph wavelets are constructed using the heat kernel, which captures the localized frequency content of a graph signal, allowing for effective feature representation on graph-structured data. To compute these basis functions efficiently, we approximate the heat kernel using a truncated series expansion, thereby providing an efficient way to compute the heat kernel and perform spectral graph wavelet analysis. The main contributions of this work can be summarized as follows:

- We propose a novel graph encoder-decoder network that effectively learns and encodes underlying patterns and relationships in graph-structured data for anomaly detection.
- We introduce an effective pooling strategy that focuses on extracting local patterns within graphs, leading to more robust and representative feature encoding.
- We incorporate a denoising operation into our network architecture using spectral graph wavelets to reduce the impact of noise during the decoding stage with the aim of further improving the quality of the reconstructed graph data.
- Through extensive experiments on six benchmark datasets, we demonstrate that our proposed method outperforms several state-of-the-art baselines in terms of various evaluation metrics.

This chapter is structured as follows: In Section 2, we review important relevant work. In Section 3, we introduce a graph encoder-decoder model with a robust pooling strategy for unsupervised anomaly detection. In Section 4, we present experimental results and ablation studies to demonstrate the competitive performance of our approach on six standard benchmarks. Finally, we conclude in Section 5 and identify promising directions for future research.

4.2 Related Work

The basic goal of anomaly detection in attributed graphs is to identify nodes in a graph that exhibit unusual or unexpected behavior based on their attributes or features. Graph pooling, on the other hand, aims to reduce the complexity and size of a graph while preserving its salient features. In this section, we summarize relevant work at the intersection of graph anomaly detection and graph pooling. This integration enables the development of robust approaches that not only detect anomalies effectively but also produce compact graph representations that capture important structural information.

Graph Anomaly Detection. The aim of graph anomaly detection is to analyze the topology and attributes of a graph to identify nodes that deviate from the expected patterns. It is a challenging task due to the inherent complexity and structural characteristics of graphs. Graph neural networks, particularly graph convolutional networks (GCNs), have recently become the method of choice for anomaly detection on graphs. These models are predominantly performed in an unsupervised manner due to the cost of acquiring anomaly labels. Li *et al.* [132] propose a graph anomaly detection framework that leverages residual analysis to identify deviations between predicted and observed attribute values in the graph. It focuses on detecting attribute-based anomalies in attributed graphs by analyzing the residuals, which are the differences between predicted attribute values and the observed attribute values. Ding *et al.* [28] introduce a GCN-based autoencoder for anomaly detection in attributed graphs by considering both topological structure and nodal attributes. It includes an attributed network encoder designed to capture both network structure and nodal attributes in order to facilitate node embedding representation learning with GCN, a structure reconstruction decoder that reconstructs the original graph topology using the learned node embeddings, and an attribute reconstruction decoder to reconstruct the observed nodal attributes based on the obtained node embeddings. Wang *et al.* [33] design a one-class classification method for graph anomaly detection by mapping the training nodes into a hypersphere in the embedding space via graph neural networks. Zhou *et al.* [34] present an abnormality-aware graph neural network, which utilizes a subtractive aggregation technique to represent each node based on its deviation from its neighbors. Nodes that

are considered normal and have high confidence are used as labels to train the network in learning a customized hypersphere criterion for identifying anomalies within the attributed graph. Pei *et al.* [35] introduce a graph anomaly detection approach that captures the sparsity and nonlinearity present in attributed graphs through the use of GCNs, learns residual information, and employs a residual-based attention mechanism to mitigate the negative impact caused by anomalous nodes. Zhuang *et al.* [36] propose a subgraph centralization approach for graph anomaly detection, addressing the weaknesses of existing detectors in terms of computational cost, suboptimal detection accuracy, and lack of explanation for identified anomalies, leading to the development of a graph-centric anomaly detection framework. Duan *et al.* [37] present a multi-view, multi-scale contrastive learning framework with subgraph-subgraph contrast for graph anomaly detection by combining various anomalous information and calculating the anomaly score for each node. However, most of these approaches do not incorporate pooling operations explicitly. Instead, they rely on simple aggregation methods like mean or max pooling to downsample the graph. In contrast, our method introduces a novel pooling strategy based on locality-constrained linear coding, which preserves local structure and captures more informative and discriminative features during the pooling process. Moreover, our method employs a graph encoder-decoder architecture, where the encoder learns high-level features from the graph data, and the decoder reconstructs the original data from these features. This design allows our model to learn more effective and compact representations of the graph, making it better suited for anomaly detection tasks.

Graph Pooling. Graph pooling is a commonly-used operation in graph neural networks, with the aim of producing a compact yet informative representation of the graph structure by summarizing the information contained in the nodes of the graph. By applying a pooling operation, the graph can be transformed into a coarse representation that is easier to analyze or use as input for downstream tasks such as graph anomaly detection. Graph pooling methods can be broadly categorized into two types: global pooling and hierarchical pooling. Global pooling methods summarize the information of all nodes in the graph into a single vector or scalar [38–40], while hierarchical pooling methods recursively apply a pooling operation to the graph, producing a hierarchy of coarser graphs with decreasing numbers of nodes [41–43]. On the other hand, spectral clustering pooling techniques consider graph pooling as a cluster assignment task [43], which categorizes nodes into a set of clusters based on their learned embeddings and constructs the coarser graph based on new nodes using a learned or predefined cluster assignment matrix. Ying *et al.* [41] propose DiffPool, a differentiable pooling strategy that can generate hierarchical representations of graphs by learning a cluster assignment matrix in an end-to-end fashion. This learned assignment matrix contains the probability values of nodes in each layer being assigned to clusters in the next layer gener-

ated based on node features and topological structure of the graph. However, DiffPool generates a dense assignment matrix, making it impracticable for large graphs. Moreover, DiffPool can be sensitive to the initial node embeddings used in the clustering process. If the initial embeddings are not well-aligned with the underlying graph structure, it can lead to inaccurate clustering and subsequent pooling, affecting the quality of the learned representations. Other hierarchical pooling methods include SAGPool [42] and gPool [44], which leverage node features and graph topology to learn hierarchical representations. To perform graph pooling, SAGPool selects the most important nodes based on their self-attention scores. The selected nodes are then retained in the pooled representation, while the remaining nodes are discarded. On the other hand, gPool samples nodes according to their scalar projection values using a trainable projection vector, resulting in a coarsened graph. However, self-attention mechanisms and the trainable projection vectors tend to heavily influence the pooling process and are sensitive to the quality of attention or projection vectors. If the attention mechanisms fail to properly capture relevant node relationships or if the trainable vectors are not adequately optimized, it can negatively impact the quality of the pooling operation. Moreover, these pooling operations introduce additional trainable parameters to obtain a coarser graph, thereby increasing the overall complexity of the model. This can lead to a higher number of parameters that need to be learned, resulting in increased memory requirements and computational overhead during training and inference. In addition, these extra trainable parameters in the pooling operations can potentially lead to overfitting, especially when the available training data is limited. In contrast to these pooling operations, our proposed pooling strategy distinguishes itself in several ways. First, it does not rely on learnable parameters, making it more flexible and adaptable. Second, it leverages locality-constrained linear coding to ensure that the local structure of the graph is preserved during the pooling process. Third, it effectively generates a coarser graph representation while preserving the crucial structural attributes that are most relevant to the graph's overall characteristics. By preserving local structure and capturing significant graph characteristics, our pooled strategy enhances the discriminative power of the proposed anomaly detection model, leading to more accurate and reliable results.

4.3 Proposed Method

In this section, we introduce a graph encoder-decoder network for reconstructing both the graph structure and node features. It leverages an encoder-decoder framework to learn and generate a representation of the original graph. By jointly reconstructing both the graph structure and node features, the proposed model can capture and preserve the intricate relationships between nodes and the underlying characteristics of the graph.

Basic Notions. An attributed graph is a graph where each node is associated with a set of attributes or features, such as demographic information, transaction history, or social connections. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be an attributed graph, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of N nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^\intercal$ an $N \times F$ feature matrix of node attributes (i.e., $\mathbf{x}_i$ is an F-dimensional row vector for node i). We denote by $\mathbf{A}$ an $N \times N$ adjacency matrix whose (i, j)-th entry is equal to 1 if i and j are neighboring nodes, and 0 otherwise. We also denote by $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ the adjacency matrix with self-added loops, where $\mathbf{I}$ is the identity matrix.

Problem Statement. The goal of unsupervised node anomaly detection in an attributed graph is to identify anomalous nodes in a graph without the use of labeled training data. In other words, there is no available ground truth information that indicates which nodes are anomalous and which ones are not. Given an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, the objective of unsupervised node anomaly detection is to learn a scoring function $s : \mathcal{V} \to \mathbb{R}$ that assigns an anomaly score to each node in the graph. Once anomaly scores are computed, the r nodes with the highest anomaly scores are selected based on a user-defined value of r. These selected nodes are then identified as anomalies. In other words, nodes with high anomaly scores are considered anomalous, while nodes with lower scores are deemed normal.

Approach Overview. The overall framework of our proposed approach is shown in Figure 4.1. The objective is to design a graph encoder-decoder model that can transform an input graph $\mathcal{G}$ into a coarser representation $\mathcal{G}'$, and then reconstruct it back. The proposed model is comprised of two main components: an encoder and a decoder. The encoder consists of a graph convolutional network, which performs convolutions on the graph, and a graph pooling later, which downsamples the graph and extract the most important information from the graph while reducing the number of parameters and computational complexity of the network. The decoder, on the other hand, consists of a graph unpooling layer, which upsamples the representation of the graph, followed by a graph deconvolutional network, which produces an output node feature representation that approximates the input node feature matrix of the original graph.

4.3.1 Encoder

In the encoding stage, a GCN encoder takes as input an adjacency matrix and a node feature matrix, and produces a latent representation of the graph that captures its structural and feature information by performing convolutions on the graph and aggregating information from neighboring nodes. Then, a graph pooling layer is applied to generate a coarser graph representation using a sparse coding based approach, resulting in a coarsened adjacency matrix and a coarsened node embedding matrix.

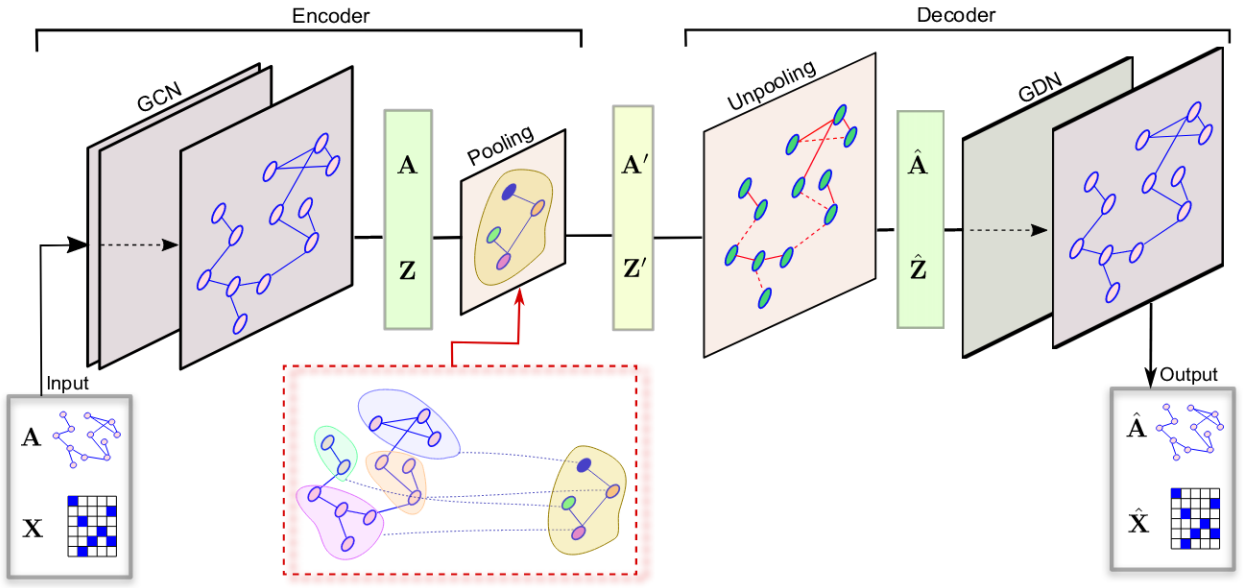


Figure 4.1: Overview of the proposed graph encoder-decoder network architecture. The model consists of two main components: an encoder and a decoder. In the encoding stage, a graph convolutional network (GCN) encoder is used to generate a latent representation, followed by a graph pooling layer to coarsen the graph. In the decoding stage, an unpooling layer is applied to the coarser graph, followed by a graph deconvolutional network (DGN) decoder to reconstruct the graph.

GCN Encoder. The basic idea of a GCN [1] is to learn a set of filters that can extract meaningful features from the graph. This is achieved by aggregating information from neighboring nodes to learn a new representation for each node. The layer-wise propagation rule of GCN is based on the graph convolution operation, which computes the output of a node based on the features of its neighboring nodes as follows:

$$\mathbf{H}^{(\ell+1)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{H}}^{(\ell)}\mathbf{W}^{(\ell)}), \quad \ell = 0, \ldots, L-1 \tag{4.1}$$

where $\mathbf{W}^{(\ell)}$ is a learnable weight matrix, $\mathbf{H}^{(\ell)} \in \mathbb{R}^{N \times F_\ell}$ is the input feature matrix of the ℓ-th layer with F_ℓ as embedding dimension, $\tilde{\mathbf{D}} = \mathrm{diag}(\tilde{\mathbf{A}}\mathbf{1})$ is the diagonal degree matrix corresponding to the adjacency matrix with self-added loops, $\mathbf{1}$ is an N-dimensional vector of all ones, and $\sigma(\cdot)$ is an element-wise activation function such as ReLU. The input of the first layer is the initial feature matrix $\mathbf{H}^{(0)} = \mathbf{X}$.

The final output node embeddings are given by an $N \times P$ feature matrix $\mathbf{Z} = \mathrm{GCN}(\mathbf{A}, \mathbf{X})$ generated at the L-layer of the GCN encoder as follows:

$$\mathbf{Z} = \mathbf{H}^{(L)} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\intercal, \tag{4.2}$$

where P is the embedding dimension at the final network layer, and $\mathbf{z}_i$ is the i-th row of $\mathbf{Z}$ representing the output embedding of node i. These learned low-dimensional embeddings capture the structural and semantic similarities of the graph nodes.

Graph Pooling Layer. The purpose of a graph pooling layer is to reduce the size and complexity of a graph while preserving its important features and structural characteristics. Given the graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ of the input graph $\mathcal{G}$ and the output node embeddings matrix $\mathbf{Z} \in \mathbb{R}^{N \times P}$ of the GCN encoder, our aim is to design a graph pooling strategy that takes the graph $\mathcal{G}$ as input and produces a coarser graph $\mathcal{G}'$ comprised of $K < N$ nodes, with a weighted adjacency matrix $\mathbf{A}' \in \mathbb{R}^{K \times K}$ and a node embedding matrix $\mathbf{Z}' \in \mathbb{R}^{K \times P}$. To generate the coarser graph $\mathcal{G}'$, we use a cluster assignment matrix obtained via locality-constrained linear coding [136], which is a variant of sparse coding that imposes a locality constraint on the weight coefficients, such that each coefficient is only allowed to depend on nearby basis functions. This constraint is enforced by adding a penalty term to the optimization problem that encourages the weight coefficients to be small for distant basis vectors and large for nearby basis vectors.

Determining the Assignment Matrix. Inspired by locality-constrained linear coding (LLC) [136], we seek to find an assignment matrix by solving a least square fitting problem with a locality regularization term. Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^\intercal$ be the embedding feature matrix obtained by the GCN encoder, where $\mathbf{z}_i$ is a P-dimensional embedding vector for node i. We denote by $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_K)^\intercal \in \mathbb{R}^{K \times P}$ a codebook (also called vocabulary) constructed via clustering by quantizing the N embedding vectors into K basis vectors. These basis vectors are usually defined as the centers of K clusters obtained via K-means clustering on the embedding feature matrix $\mathbf{Z}$, where $\mathbf{v}_k$ is a P-dimensional vector associated to cluster k.

In order to perform the pooling operation, we seek to find a cluster assignment matrix $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_N)^\intercal \in \mathbb{R}^{N \times K}$ via LLC, where each code $\mathbf{u}_i$ is a K-dimensional vector obtained by solving the following regularized least-squares problem:

$$\mathbf{u}_i = \arg \min_{\mathbf{u}_i \mathbf{1}=1} \|\mathbf{z}_i - \mathbf{u}_i \mathbf{V}\|^2 + \lambda \|\mathbf{d}_i \odot \mathbf{u}_i\|^2, \tag{4.3}$$

where λ is a regularization hyperparameter, $\odot$ denotes the element-wise multiplication, $\mathbf{d}_i$ is a K-dimensional vector defined as

$$\mathbf{d}_i = (\exp(\|\mathbf{z}_i - \mathbf{v}_1\|/\delta), \ldots, \exp(\|\mathbf{z}_i - \mathbf{v}_K\|/\delta)), \tag{4.4}$$

which measures the similarity between the i-th embedding and all basis vectors in the codebook, and δ is a parameter to adjust the weight decay speed for the locality adaptor. Note that the elements of the cluster assignment matrix represent the weights of the basis vectors that are used

to reconstruct the embedding feature matrix, while enforcing locality regularization. It should also be noted that the LLC code $\mathbf{u}_i$ is sparse in the sense that it only comprises a few significant values.

In practice, an approximated LLC is employed for fast encoding by removing the regularization term (i.e., locality constraint) and using the R nearest neighbors of $\mathbf{z}_i$ as a set of basis vectors, thereby reducing the computational complexity from $\mathcal{O}(K^2)$ to $\mathcal{O}(K+R^2)$, where K is the number of basis vectors in the vocabulary and $R \ll K$. Since the value of K is typically small, the LLC algorithm can be executed quickly in practice.

Finally, we generate the assignment matrix $\mathbf{S}$ as follows:

$$\mathbf{S} = \text{softmax}(\mathbf{U}), \tag{4.5}$$

where the softmax function is applied row-wise. The i-th row of the cluster assignment matrix $\mathbf{S} \in \mathbb{R}^{N \times K}$ represents the probabilities of node i to be assigned to each of the K clusters, and each column represents a cluster.

Pooling Strategy with Assignment Matrix. The cluster assignment matrix assigns each graph node to a specific cluster, and plays an important role in determining the new representation of the coarser graph produced by the graph pooling operation [41]. The coarsening process aims to reduce the size of a graph by grouping nodes into clusters, while preserving the most important structural features of the graph. Specifically, given the adjacency matrix $\mathbf{A}$ and node embedding matrix $\mathbf{Z}$ of the input graph, we define the locality-constrained pooling (LCPool) operator or layer as follows:

$$(\mathbf{A}', \mathbf{Z}') = \text{LCPool}(\mathbf{A}, \mathbf{Z}), \tag{4.6}$$

where

$$\mathbf{A}' = \mathbf{S}^\intercal \tilde{\mathbf{A}} \mathbf{S} \in \mathbb{R}^{K \times K} \quad \text{and} \quad \mathbf{Z}' = \mathbf{S}^\intercal \mathbf{Z} \in \mathbb{R}^{K \times P} \tag{4.7}$$

are the adjacency matrix of the coarser graph and its new matrix of node embeddings, respectively. Note that $\mathbf{A}'$ is a weighted adjacency matrix representing the connectivity of the clusters. Each row and column of this coarsened adjacency matrix represents a cluster of nodes, while its (i, j)-th element represents the connectivity strength between cluster i and cluster j. Similarly, the (i, j)-th element of the new matrix of embeddings $\mathbf{Z}'$ can be viewed as a weighted sum of the elements of the j-th column of $\mathbf{Z}$, where the weights are given by the corresponding elements of the i-th row of $\mathbf{S}$ (i.e., probabilities of node i to be assigned to each cluster of the K clusters).

4.3.2 Decoder

In the decoding stage, we employ a graph unpooling layer, which upsamples the representation of the graph, followed by a graph deconvolutional network (GDN) decoder, which reconstructs the

node feature matrix of the original graph. The aim of the unpooling layer is to upsample the graph by mapping the coarser graph structural and feature representations to finer ones. It basically performs the inverse operation of the pooling layer. Specifically, the unpooling operation attempts to reconstruct the original graph structure and nodal features from the coarser graph representation obtained after pooling. By applying the unpooling operation, the model can reconstruct the finer details of the original graph that may have been lost during the pooling process. This allows for a more accurate representation of the graph's structure and nodal features. Also, the unpooling operation can help provide a clearer understanding of the graph by recovering the original graph structure. This allows for better interpretability and analysis of the graph's properties and characteristics. By reconstructing the original graph, the unpooling operation aims to retain the most significant structural characteristics and nodal features. This can be valuable in applications where preserving important information is crucial, such as anomaly detection. The GDN decoder, on the other hand, is used to decode or reconstruct the original graph from the finer representations generated by the unpooling layer.

Graph Unpooling Layer. The purpose of the unpooling layer is to reconstruct the original graph structure from the down-sampled feature maps produced by the pooling layer. One advantage of the coarsened adjacency matrix $\mathbf{A}'$ is that it preserves the most important structural features of the original graph. To reconstruct the graph structure and nodal features of the original graph, we use a locality-constrained unpooling (LCUnpool) strategy, which takes a coarser graph with features as input and produces a finer representation of the original graph, incorporating both the desired structure and features. We define the LCUnpool operator or layer as follows:

$$(\hat{\mathbf{A}}, \hat{\mathbf{Z}}) = \text{LCUnpool}(\mathbf{A}', \mathbf{Z}'), \tag{4.8}$$

where

$$\hat{\mathbf{A}} = \mathbf{S}\tilde{\mathbf{A}}'\mathbf{S}^\intercal \in \mathbb{R}^{N \times N} \quad \text{and} \quad \hat{\mathbf{Z}} = \mathbf{S}\mathbf{Z}' \in \mathbb{R}^{N \times P} \tag{4.9}$$

are the unpooled adjacency matrix and the unpooled matrix of node embeddings, respectively. Note that this graph unpooling strategy aggregates information from the node neighborhood by leveraging the normalized coarsened adjacency matrix $\tilde{\mathbf{A}}'$, which takes into account the relative importance of each node in the graph.

GDN Decoder. The deconvolution process can be thought of as an inverse operation to graph convolution, with the aim of recovering the original node features. The key idea behind the GDN decoder is to use a learnable deconvolution operation to reverse the convolutional transformation applied by the GCN encoder. This deconvolution operation needs to take into account the graph structure and ensure that the reconstructed features are consistent with the graph topology. Similar

to [60, 137], we apply a deconvolution operation by taking the unpooled adjacency matrix $\hat{\mathbf{A}}$ and node representation $\hat{\mathbf{Z}}$ as input for the GDN encoder, yielding a reconstructed node feature matrix $\mathbf{H} \in \mathbb{R}^{N \times F}$ given by

$$\mathbf{H} = \sigma((\mathbf{I} + \hat{\mathbf{L}})\hat{\mathbf{Z}}\mathbf{W}), \tag{4.10}$$

where $\hat{\mathbf{L}} = \mathbf{I} - \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ is the normalized Laplacian matrix, $\hat{\mathbf{D}} = \mathrm{diag}(\hat{\mathbf{A}}\mathbf{1})$ is the diagonal degree matrix, and $\mathbf{W} \in \mathbb{R}^{P \times F}$ is a learnable weight matrix.

The deconvolution operation can be seen as a graph diffusion process that spreads the convolutional features back to their original locations, while taking into account the underlying graph structure. However, the graph convolution of the GCN encoder can be interpreted as a special form of Laplacian smoothing [59], which is a graph filtering operation that can be viewed as a low-pass filter that removes high-frequency noise. Hence, applying the inverse operation of the graph convolution may introduce undesirable noise into the output graph. This issue can be remedied using spectral graph wavelet denoising [137], a graph signal processing technique that aims to remove noise from a graph signal by leveraging a set of wavelet functions, which are usually defined as a set of filters operating on the graph Laplacian eigenvalues.

The advantage of using spectral graph wavelets for denoising is that it is possible to remove noise that corresponds to high-frequency components of the signal, while preserving the low-frequency components that carry the main information of the signal. Moreover, they provide a flexible and adaptive framework for capturing the underlying structure and smoothness of the graph signal. Specifically, let $\hat{\mathbf{L}} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{\intercal}$ be an eigendecomposition of the normalized Laplacian matrix, where $\mathbf{\Phi}$ is a matrix whose columns are the eigenvectors (i.e., graph Fourier basis) and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ is a diagonal matrix comprised of the corresponding eigenvalues. Spectral graph wavelets have shown to allow localization of graph signals in both spatial and spectral domains [111, 138, 139]. Let $g_s(\lambda) = e^{-\lambda s}$ be the transfer function (also called frequency response) of the heat kernel with scaling parameter s. The spectral graph wavelet basis $\mathbf{\Psi}_s$ is defined as

$$\mathbf{\Psi}_s = \mathbf{\Phi}\mathbf{G}_s\mathbf{\Phi}^{\intercal}, \tag{4.11}$$

where

$$\mathbf{G}_s = g_s(\mathbf{\Lambda}) = \mathrm{diag}(g_s(\lambda_1), \ldots, g_s(\lambda_N)) \tag{4.12}$$

is a diagonal matrix of transformed eigenvalues via the transfer function. Note that $\mathbf{\Psi}_s$ is also referred to as the heat kernel matrix whose inverse $\mathbf{\Psi}_s^{-1}$ is obtained by simply replacing the scale parameter s with its negative value. The spectral graph wavelet basis and its inverse can also be computed efficiently using polynomial approximations via the Maclaurin series, which can be

used to approximate the heat kernel on a graph, by expanding it as a polynomial in the normalized Laplacian matrix, and then truncating the series at a finite order [137].

Therefore, using the feature representation matrix $\mathbf{H}$ and both the spectral graph wavelet basis and its inverse, the reconstructed node feature matrix $\hat{\mathbf{X}} = \text{GDN}(\hat{\mathbf{A}}, \hat{\mathbf{Z}})$ via the GDN decoder can be obtained as follows:

$$\hat{\mathbf{X}} = \mathbf{\Psi}_t \text{ReLU}(\mathbf{\Psi}_t^{-1}\mathbf{H}\mathbf{W}_1)\mathbf{W}_2, \tag{4.13}$$

where $\mathbf{W}_1 \in \mathbb{R}^{F\times P}$ and $\mathbf{W}_2 \in \mathbb{R}^{P\times F}$ are learnable weight matrices. The reconstructed node feature matrix aims to provide an approximation of the original node feature matrix. By reconstructing the node feature matrix, we can gain insights into the estimated values of node features, understand the patterns within the graph, and utilize this information for further downstream tasks such as anomaly detection.

4.3.3 Model Training

In order to detect anomalies, we minimize the joint reconstruction loss of the nodal attributes and topological structure, which allows us to learn the reconstruction errors. This loss function is defined as a weighted combination of the structure reconstruction error and the feature reconstruction error

$$\mathcal{L} = (1-\alpha)\|\mathbf{A} - \hat{\mathbf{A}}\|_F^2 + \alpha\|\mathbf{X} - \hat{\mathbf{X}}\|_F^2, \tag{4.14}$$

where $\|\cdot\|_F$ denotes the Frobenius norm and α is a weighting hyperparameter that controls the weight or importance given to each error term. The structure reconstruction loss quantifies how well our model approximates the original adjacency matrix, while the feature reconstruction loss measures the quality of the reconstructed node feature representation.

Our proposed graph encoder-decoder model can iteratively approximate the input graph by minimizing the loss function $\mathcal{L}$ to learn the weight matrices. The goal is to adjust the model's parameters such that the loss function is minimized during training. This is typically achieved using a stochastic gradient descent optimizer, which iteratively adjusts the parameters based on the gradient of the loss function with respect to the model parameters. We usually stop training when the performance of the model no longer improves after a certain number of iterations or epochs. The final reconstruction errors are then used to compute the anomaly score of node i as follows:

$$\text{s}_i = (1-\alpha)\|\mathbf{a}_i - \hat{\mathbf{a}}_i\| + \alpha\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|, \tag{4.15}$$

which is defined as a weighted sum of a structure error term and a feature error term, where $\mathbf{a}_i$ and $\mathbf{x}_i$ are the i-th rows of $\mathbf{A}$ and $\mathbf{X}$ representing the structure and feature vectors of node i, while $\hat{\mathbf{a}}_i$ and $\hat{\mathbf{x}}_i$ are the i-th rows of $\hat{\mathbf{A}}$ and $\hat{\mathbf{X}}$ representing the recovered structure and feature vectors. The

nodes are sorted in descending order according to their anomaly scores, and the nodes with the highest anomaly scores are identified as anomalies. Note that a high value of the structure error term implies that the i-th node in the graph is more likely to be an anomaly based on the graph structure, while a high value of the feature error term indicates an anomalous node from the feature perspective.

4.4 Experiments

In this section, we present our experimental setup and empirical results. Our aim is to assess the effectiveness and performance of the proposed model in comparison with state-of-the-art methods for graph anomaly detection.

4.4.1 Experimental Setup

Datasets. We evaluate the performance of our method against state-of-the-art approaches on two groups of standard benchmark datasets:

Citation networks: Cora, Citeseer, Pubmed [116], and ACM [140] are citation network datasets, which are publicly available and consist of scientific publications. In these networks, nodes denote published articles and edges represent the citation relationships between articles. Each node is described by a binary feature vector indicating the absence/presence of the corresponding word from the dictionary.

Social networks: BlogCatalog and Flickr [141] are two typical social network datasets acquired from the blog sharing website, BlogCatalog, and the image hosting and sharing website, Flickr, respectively. In these datasets, nodes represent users of websites and links represent the relationships between users. Social network users typically create personalized content, such as blog posts or photo sharing with tag descriptions, which are considered as attributes of the nodes.

We follow the same preprocessing procedure from [28, 142, 143]. Since there is no ground-truth of anomalous nodes, it is necessary to artificially introduce synthetic anomalies into the clean attributed networks for the purpose of evaluation. To accomplish this, a collection of anomalies, including both structural and contextual anomalies, are injected into each dataset. Dataset statistics are summarized in Table 4.1.

Baselines. To demonstrate the effectiveness of our method, we include strong baselines for anomaly detection, including local outlier factor (LOF) [144], structural clustering algorithm for networks (SCAN) [145], anomaly mining of entity neighborhoods (AMEN) [131], residual analysis for anomaly detection in attributed networks (Radar) [132], a joint modeling approach

Table 4.1: Summary statistics of datasets.

Dataset	Nodes	Edges	Features	Anomalies
BlogCatalog	5196	171743	8189	300
Flickr	7575	239738	12407	450
ACM	16484	71980	8337	600
Cora	2708	5429	1433	150
Citeseer	3327	4732	3703	150
Pubmed	19717	44338	500	600

for anomaly detection on attributed networks (ANOMALOUS) [133], deep anomaly detection on attributed networks (a\Dominant) [28], deep graph infomax (DGI) [146], contrastive self-supervised learning framework for anomaly detection (CoLA) [142], abnormality-aware graph neural network (AAGNN) [34], one class graph neural network (OCGNN) [33], Graph Deviation Networks (**GDN!**) [147], DGAE-GAN [148], Residual Graph Convolutional Network (ResGCN) [35]. For baselines, we mainly consider methods that are closely related to our proposed approach and/or the ones that are state-of-the-art graph anomaly detection techniques.

Evaluation Metrics. To evaluate the performance of our proposed model against the baseline methods, we adopt several evaluation metrics, including AUC, Precision@K, Recall@K, F1@K, and NDCG@K. AUC summarizes the information contained in the ROC curve, plotting the true positive rate vs. false positive rate at various thresholds. Larger AUC values indicate better performance at distinguishing between anomalous and normal nodes. Considering the list of nodes sorted based on the anomaly score, Precision@K focuses on the proportion of true anomalous nodes, which are included in the top-K position of ranked nodes. Recall@K measures the proportion of known anomalous nodes selected out of all ground-truth anomalies. F1@$K$is the harmonic mean of Precision and Recall. NDCG@K is a measure of ranking quality, which provides a weighted score that favors rankings where anomalous nodes are ranked closer to the top. These evaluation metrics provide insights into the effectiveness of the method in identifying anomalies and distinguishing them from normal instances.

Implementation Details. We implement our model in PyTorch, and train it using Adam [149] optimizer on the BlogCatalog, Flickr, and ACM datasets for 300 epochs, and on the Cora, Citeseer, and Pubmed datasets for 100 epochs. The learning rate for BlogCatalog, Flicker, ACM, and Cora is set to 10^{-4}, while the learning rates for Citeseer and Pubmed are set to 10^{-5} and 10^{-3}, respectively. For the GCN encoder, we set the number of hidden layers to 3. The embedding dimension is set to 512 for Cora, Citeseer and Pubmed, and to 218 for BlogCatalog, Flicker and ACM. In all

experiments, we set the scaling parameter s to 1 in the wavelet bases. For the approximated LLC, we set the number of neighbors R to 5. A reasonable choice for the weighting hyperparameter α of the loss function is between 0.5 and 0.8 for all datasets. All other hyperparameters and initialization strategies are those suggested by the baselines' authors. We tune hyper-parameters using the validation set, and terminate training if the validation loss does not decrease after 10 consecutive epochs.

4.4.2 Anomaly Detection Performance

We evaluate the anomaly detection performance of our approach against strong baseline methods. Table 4.2 reports the AUC scores for our model and baselines on the six benchmark datasets. The AUC scores for the baseline methods on the citation networks are taken from [28] and [142]. The best results are shown in bold, and the second best results are underlined. As can be seen, our method outperforms the baselines on most datasets with relative improvements of 1.23%, 0.16%, 0.76% and 1.85% in terms of AUC on BlogCatalog, Flickr, Cora and Citeseer, respectively.

Table 4.2: Test AUC (%) scores on four citation networks and two social networks. Boldface numbers indicate the best performance, whereas the underlined numbers indicate the second best performance.

Method	BlogCatalog	Flickr	ACM	Cora	Citeseer	Pubmed
LOF [144]	49.15	48.81	47.38	-	-	-
SCAN [145]	27.27	26.86	35.99	-	-	-
AMEN [131]	66.48	60.47	53.37	62.66	61.54	77.13
Radar [132]	71.04	72.86	69.36	65.87	67.09	62.33
Anomalous [133]	72.81	71.59	71.85	57.70	63.07	73.16
Dominant [28]	78.13	74.90	74.94	81.55	82.51	80.81
DGI [146]	58.27	62.37	62.40	75.11	82.93	69.62
CoLA [142]	78.54	75.13	82.37	<u>87.79</u>	<u>89.68</u>	**95.12**
OCGNN [33]	55.50	48.91	50.00	86.97	85.62	74.72
GDN [147]	54.24	52.40	69.15	75.77	78.89	69.15
AAGNN [34]	<u>81.84</u>	<u>82.99</u>	**85.64**	-	-	-
DGAE-GAN [148]	81.80	79.50	83.80	-	-	-
ResGCN [35]	78.50	78.00	76.80	84.79	76.47	80.79
Ours	**82.85**	**83.12**	<u>84.69</u>	**88.46**	**91.34**	<u>92.81</u>

In Tables 4.3, 4.4 and 4.5, we report the results in terms of Precision@K, Recall@K and F1@K scores, respectively, on all datasets for various values of K ranging from 50 to 300. As can be seen, the shallow methods such as LOF, SCAN, AMEN, Radar and ANOMALOUS do not show a competitive anomaly detection performance. This is largely attributed to the fact that their mechanisms

have limited capability to detect anomalous nodes in graph-structured data with high-dimensional features and/or complex structures. For instance, both LOF and SCAN yield unsatisfactory results due in large part to the fact that they perform anomaly detection without any knowledge about nodal attributes or graph structure. Among the baselines that consider both attributes and structure, AMEN focuses on finding anomalous connected subgraphs rather than nodes, resulting in poor performance. The residual analysis based models, Radar and Anomalous, show superior performance over the conventional anomaly detection methods (LOF, SCAN and AMEN). However, they can only capture the linear dependency because they are based on matrix factorization. Compared to the other deep learning baselines, our proposed anomaly detection model shows a stronger detection performance and generalization ability.

Table 4.3: Test Precision@K (%) scores of our approach and baselines on four citation networks and two social networks. Boldface numbers indicate the best performance, whereas the underlined numbers indicate the second best performance.

	BlogCatalog				Flicker				ACM				Cora				Citeseer				Pubmed			
K	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300
LOF [144]	30.0	22.0	18.0	18.3	42.0	38.0	27.0	23.7	6.0	6.0	4.5	3.7	-	-	-	-	-	-	-	-	-	-	-	-
Radar [132]	66.0	67.0	55.0	41.6	74.0	70.0	63.5	50.3	5.6	5.8	5.2	4.3	-	-	-	-	-	-	-	-	-	-	-	-
Anomalous [133]	64.0	65.0	51.5	41.7	79.0	71.0	65.0	51.0	60.0	57.0	51.0	41.0	-	-	-	-	-	-	-	-	-	-	-	-
AMEN [131]	60.0	58.0	49.6	38.3	67.0	64.0	55.0	46.1	52.0	49.0	43.2	36.0	54.6	47.2	29.0	23.0	64.0	44.0	23.0	21.6	56.0	54.1	49.0	45.7
Dominant [28]	76.0	71.0	59.0	**47.0**	77.0	73.0	68.5	59.3	62.0	59.0	54.0	**49.7**	68.0	**55.0**	36.0	27.0	76.0	51.0	32.0	25.3	70.0	66.0	63.0	56.0
DGI [147]	52.0	51.0	43.6	32.3	59.0	57.7	46.0	45.4	46.0	41.4	38.0	35.4	47.1	39.0	25.3	19.1	54.0	36.3	21.0	17.9	49.0	48.0	44.0	39.5
CoLA [142]	62.0	58.0	39.5	31.0	60.0	51.0	31.5	26.7	**88.0**	**71.0**	57.5	46.8	66.0	54.0	**41.5**	**34.3**	58.0	47.0	**39.0**	**31.7**	**76.0**	69.0	58.5	55.7
Ours	**80.0**	**75.0**	**60.0**	31.7	**84.0**	**79.0**	**71.0**	**63.3**	**88.0**	69.1	**58.0**	47.1	**74.0**	**55.0**	31.0	26.0	**78.0**	**59.0**	32.5	27.3	75.8	**69.2**	**64.3**	**57.0**

Table 4.4: Test Recall@K (%) scores of our approach and baselines on four citation networks and two social networks. Boldface numbers indicate the best performance, whereas the underlined numbers indicate the second best performance.

	BlogCatalog				Flicker				ACM				Cora				Citeseer				Pubmed			
K	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300
LOF [144]	5.0	7.3	12.0	18.3	4.7	8.4	12.0	15.8	0.5	1.0	1.5	1.8	-	-	-	-	-	-	-	-	-	-	-	-
Radar [132]	11.0	22.3	36.7	41.6	8.2	15.6	28.2	33.6	4.7	9.7	17.3	21.5	-	-	-	-	-	-	-	-	-	-	-	-
Anomalous [133]	10.7	21.7	34.3	41.7	8.7	15.8	28.9	34.0	5.0	9.5	17.0	20.5	-	-	-	-	-	-	-	-	-	-	-	-
AMEN [131]	9.7	19.6	32.4	38.9	7.2	14.3	26.9	32.1	4.5	7.9	15.9	17.1	20.9	31.4	42.4	47.6	21.6	30.1	37.7	44.7	4.4	8.1	15.5	22.2
Dominant [28]	12.7	23.7	39.3	47.0	8.4	16.2	30.4	39.6	5.2	9.8	18.0	24.8	23.7	36.7	48.0	54.0	25.3	34.0	42.7	50.7	**28.3**	11.0	21.0	28.0
DGI [147]	8.4	17.1	28.2	34.1	7.3	13.0	24.4	29.3	4.3	8.4	13.7	16.5	18.1	27.1	31.9	35.8	17.2	24.1	30.3	35.8	3.5	6.6	12.1	17.7
CoLA [142]	10.4	19.5	26.5	31.2	6.7	11.5	14.1	18.0	7.3	11.9	19.3	23.4	22.0	36.0	55.3	68.7	19.3	31.3	52.0	63.3	6.3	11.5	19.5	27.9
Ours	**24.8**	**25.2**	**40.3**	**60.9**	**18.2**	**20.6**	**41.3**	**48.8**	**9.8**	**18.5**	**21.0**	**28.3**	**42.5**	**51.7**	**71.3**	**88.5**	**31.0**	**52.2**	**57.5**	**70.8**	11.8	**15.8**	**27.8**	**30.8**

Model Efficiency. The training time of our model depends on the complexity of the graph and the size of the dataset, as well as the number of layers and learnable parameters in the model. Applying

Table 4.5: Test F1@K (%) scores of our approach and baselines on four citation networks and two social networks. Boldface numbers indicate the best performance, whereas the underlined numbers indicate the second best performance.

	BlogCatalog				Flicker				ACM				Cora				Citeseer				Pubmed			
K	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300	50	100	200	300
LOF [144]	8.6	10.9	14.4	18.3	8.4	13.7	16.6	18.9	0.9	1.7	2.2	2.4	-	-	-	-	-	-	-	-	-	-	-	-
Radar [132]	18.8	33.4	44.0	41.6	14.7	25.5	39.0	40.3	5.1	7.2	7.9	7.2	-	-	-	-	-	-	-	-	-	-	-	-
Anomalous [133]	18.3	32.5	41.2	41.7	15.7	25.8	40.0	40.8	9.2	16.3	25.5	27.3	-	-	-	-	-	-	-	-	-	-	-	-
AMEN [131]	16.7	29.3	39.2	38.6	13.0	23.4	36.1	37.8	8.3	13.6	23.2	23.2	30.2	37.7	34.4	31.0	32.3	35.7	28.6	29.1	8.2	14.1	23.6	29.9
Dominant [28]	21.8	35.5	47.2	**47.0**	15.1	26.5	42.1	47.5	9.6	16.8	27.0	33.1	35.1	44.0	41.1	36.0	38.0	40.8	36.6	33.8	40.3	18.9	31.5	37.3
DGI [147]	14.5	25.6	34.2	33.2	13.0	21.2	31.9	35.6	7.9	14.0	20.1	22.5	26.2	32.0	28.2	24.9	26.1	29.0	24.8	23.9	6.5	11.6	19.0	24.4
CoLA [142]	17.8	29.2	31.7	31.1	12.1	18.8	19.5	21.5	13.5	20.4	28.9	31.2	33.0	43.2	**47.4**	**45.8**	29.0	37.6	**44.6**	**42.2**	11.6	19.7	29.3	37.2
Ours	**37.9**	**37.7**	**48.2**	41.7	**29.9**	**32.7**	**52.2**	**55.1**	**17.6**	**29.2**	**30.8**	**35.4**	**54.0**	**53.3**	43.2	40.2	**44.4**	**55.4**	41.5	39.4	**20.4**	**25.7**	**38.8**	**40.0**

LCPool in the proposed method does introduce additional computation to compute the assignment matrix. However, in practice, we observed that LCPool does not significantly increase the running time of the model. This is because each LCPool layer effectively reduces the size of the graph by creating a coarser representation of the graph, which leads to a speed-up in the subsequent graph convolution operation in the next layer. The reduction in graph size achieved by LCPool means that the graph convolutional operation in the subsequent layers processes fewer nodes and edges, resulting in a more efficient computation. As a result, any additional overhead from the computation of the assignment matrix is offset by the reduced computation in the subsequent layers. Moreover, LCPool's locality-constrained linear coding mechanism efficiently captures local patterns and generates more compact and informative embeddings, further contributing to the efficiency of the overall model. In addition, once our model is trained, the inference or anomaly detection time for a given graph is relatively fast. The model only needs to perform a forward pass through the encoder and decoder to compute the reconstruction loss and anomaly scores.

4.4.3 Parameter Sensitivity Analysis

The hyperparameter K, which is the new dimension of the embedded graph after applying graph pooling, plays an important role in the anomaly detection performance of the proposed model. We conduct a sensitivity analysis to investigate how the performance of our approach changes as we vary this embedding dimension. In Figure 4.2, we analyze the effect of this hyperparameter by plotting the ROC curves for our model on all datasets, where K varies in the set $\{100, 200, 300, 400\}$. We can see that our model generally benefits from relatively larger values of K. For all datasets, our model achieves good performance when $K = 400$.

We also analyze the effect of the trade-off hyperparameter α on model performance, and the

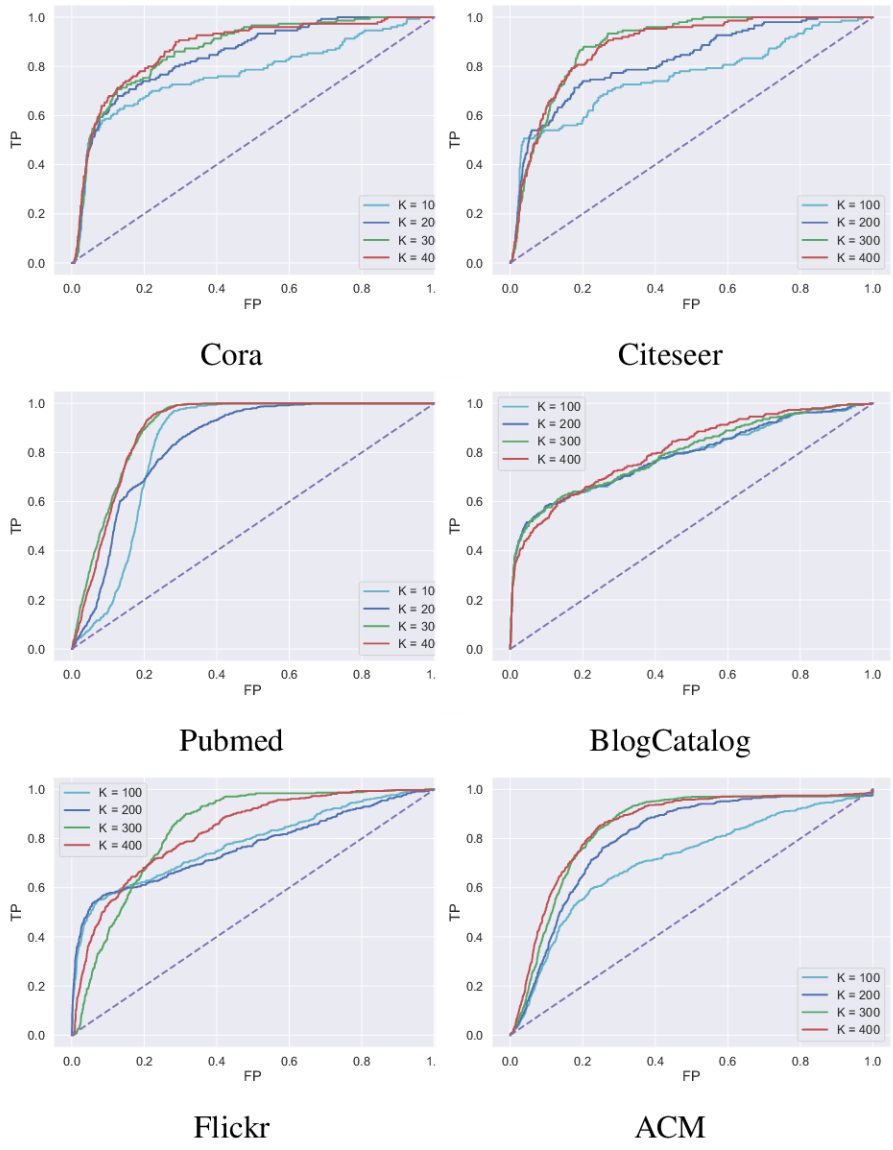


Figure 4.2: Effect of hyperparameter K on anomaly detection performance of our model using ROC curves.

results are shown in Figure 4.3. When $\alpha = 0$, our loss function reduces to the structure reconstruction loss and when $\alpha = 1$, it reduces to the feature reconstruction loss. The structure reconstruction loss evaluates the extent to which our model accurately represents the original adjacency matrix, whereas the feature reconstruction loss assesses the fidelity of the reconstructed node feature representation. As can be observed in Figure 4.3, our model generally yields higher AUC values when α is between 0.5 and 0.8. Therefore, the best detection performance is typically achieved by simultaneously considering the reconstruction errors of both graph structure and node features. Notice that in some datasets such as Flicker and BlogCatalog, assigning higher weight to the re-

construction error of the topological structure results in better performance than the reconstruction error of nodal attributes. For the other datasets, giving more weight to the reconstruction error of node features yields better results. This suggests that using node features and graph structure is vital to the model performance in identifying anomalies on data-structured data, allowing for a more accurate and comprehensive understanding of abnormal behavior in graphs. While the graph structure provides a global view of the graph topology, the node features offer a more localized perspective, capturing fine-grained details about individual nodes. Integrating both aspects enables our anomaly detection model to capture anomalies that might be missed by considering only one type of information. In fact, anomalies that exhibit complex patterns, which cannot be solely captured by either structure or features alone, can be better detected when both aspects are taken into account.

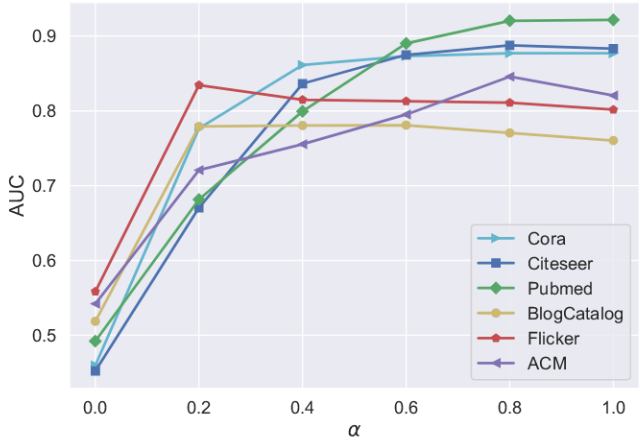


Figure 4.3: Effect of hyperparameter α on anomaly detection performance of our model using AUC as evaluation metric.

4.4.4 Ablation Study

Since there are several components in our proposed model architecture, we analyze their impacts via an ablation study. Specifically, we investigate the effectiveness of the pooling and denoising operations on the anomaly detection performance of our model. To this end, we conduct experiments for ablation studies by removing components individually. The details of our experiments are reported in Table 4.6 in terms of the AUC metric on six benchmark datasets. As can be seen, the removal of each component leads to a drop in performance. To explore the effect of the denoising operation using spectral graph wavelets, we remove it from our model framework and report the AUC results in Table 4.6. This shows that incorporating a denoising operation to our model helps

remove the amplified noise introduced during the decoding stage. The denoising operation uses a ReLU function and a trainable parameter matrix to linearly transform the noise, keeping the useful coefficients that are above zero and eliminating the noise that is below zero. This helps improve the model's performance by reducing the effect of the amplified noise on the reconstructed node feature matrix.

On the other hand, the significance of the pooling operation becomes evident when it is excluded from the proposed model architecture. This omission results in a substantial decrease in AUC values, as demonstrated in Table 4.6. The effectiveness of our pooling operation in detecting anomalies in graph data is largely attributed to the fact that locality-constrained linear coding focuses on capturing local patterns within the graph, thereby extracting more discriminative features that are specific to the local context, which is often crucial for detecting anomalies. Moreover, localized representation allows our model to better capture the subtle variations and deviations that signify anomalous behavior. In addition, LCPool incorporates the concept of locality, which encourages the extraction of robust features.

Table 4.6: Ablation analysis (AUC (%)) on six datasets. Performance better than the default version is boldfaced.

Method	BlogCatalog	Flicker	ACM	Cora	Citeseer	Pubmed
w/o denoising	80.01	76.58	74.21	75.12	83.86	82.24
w/o pooling	78.79	75.87	77.10	77.22	78.56	76.23
Ours	**82.85**	**83.12**	**84.69**	**88.46**	**91.34**	**92.81**

4.4.5 Discussions

The better performance of our proposed method is largely attributed to the combination of an effective graph encoder-decoder architecture, the utilization of LCPool for capturing local patterns, the denoising operation, and the integration of graph structure and node features. The graph encoder-decoder architecture empowers our model to learn more effective and discriminative representations of the graph-structured data. This enables it to better capture the underlying characteristics and structures of the data, aiding in the identification of anomalies. The adoption of LCPool enables our model to focus on local information, leading to the extraction of more robust and representative features. This local focus is crucial for accurately identifying anomalies and distinguishing them from normal graph nodes, especially in scenarios where anomalies exhibit complex and subtle patterns. The denoising operation, facilitated by spectral graph wavelets, mitigates the

impact of noise during the decoding stage, resulting in improved reconstruction of the node feature matrix and enhanced anomaly detection performance. Moreover, the integration of both graph structure and node features in the encoding-decoding process provides a more comprehensive view of the graph, allowing our model to capture complex patterns and correlations between nodes and their features. This holistic understanding of the graph data further bolsters the model's anomaly detection capabilities. While our model has demonstrated strong anomaly detection capabilities, there are still some limitations that warrant consideration. For instance, the model's performance may be influenced by the complexity of anomalies present in the data. Different types of anomalies may require specific adaptations or additional mechanisms for improved detection. Also, generalizing the proposed method to completely different domains or highly specialized graph data remains to be fully explored. Overall, our method presents an effective approach to graph-based anomaly detection, benefiting from the interplay of various components that collectively contribute to its superior performance. However, further exploration is necessary to address the identified limitations and fully realize the model's potential in diverse anomaly detection tasks.

CHAPTER 5

Conclusions and Future Work

This thesis has introduced innovative solutions to address semi-supervised learning challenges in graph node classification, semi-supervised anomaly detection, and unsupervised anomaly detection scenarios. Our approach for semi-supervised learning in graph node classification is based on a nonlinear function that captures informative node features while effectively mitigating over-smoothing. Inspired by the successful application of anisotropic diffusion in image and geometry processing, we leverage local graph structure and node features to learn nonlinear representations. Furthermore, we have developed a graph convolutional network enhanced with skip connections for semi-supervised anomaly detection. This model comprises a graph convolution module responsible for aggregating information from immediate node neighbors and a skip connection module that harmonizes neighborhood representations across different layers. Additionally, we have proposed an unsupervised graph encoder-decoder model to identify abnormal nodes within graphs. This is achieved through the learning of an anomaly scoring function that ranks nodes based on their degree of abnormality. In the encoding phase, we introduce an innovative pooling mechanism named LCPool, which employs locality-constrained linear coding for feature encoding. LCPool determines a cluster assignment matrix by solving a least-squares optimization problem with a locality regularization component. Section 5.1 provides a summary of the contributions made in each of the preceding chapters and outlines the conclusions drawn from the research conducted. In Section 5.1, we summarize the contributions made in each of the previous chapters, as well as the conclusions obtained from the associated research work. In Section 5.2, we discuss the limitations of the proposed approaches. Finally, we point out future research directions related to this thesis in Section 5.3.

5.1 Contributions of the Thesis

5.1.1 Anisotropic Graph Convolutional Network for Semi-supervised Learning

In Chapter 2, we introduced an anisotropic graph convolutional network for semi-supervised node classification on graph-structured data by learning efficient representations in an end-to-end fashion. We incorporated a nonlinear smoothness term into the feature diffusion rule of the convolutional neural network in a bid to tackle the issues of oversmoothing and shrinking effect. We demonstrated through extensive experimental results the competitive or superior performance of AGCN in terms of classification accuracy over standard baseline methods on several benchmarks, including citation networks and image datasets. We also showed that AGCN can be integrated into existing graph-based convolutional networks for semi-supervised learning using both co-training and self-training. In addition, we performed a statistical analysis using analysis of variance and pairwise multiple comparison, showing that the performance of our model is better or comparable with the baselines.

5.1.2 Graph Fairing Convolutional Networks for Anomaly Detection

In Chapter 3, we introduced a graph convolutional network with skip connection for semi-supervised anomaly detection on graph-structured data by learning effective node representations in an end-to-end fashion. The update rule of the proposed graph fairing convolutional network (GFCN) is theoretically motivated by implicit fairing and derived directly from the Jacobi iterative method. GFCN integrates skip connections between the initial feature matrix and each hidden layer. This allows our model to retain and reuse the original node features throughout the network, enabling better information propagation. We also showed that GFCN has the same time and memory complexity as the standard GCN, despite the inclusion of skip connections for improved node representations. Through extensive experiments, we demonstrated the competitive or superior performance of our model in comparison with the current state of the art on five benchmark datasets.While GFCN's intuitive design provides a solid theoretical foundation, it may face scalability challenges when dealing with very large graphs like many GCN-based methods.

5.1.3 A Graph Encoder-Decoder Network for Unsupervised Anomaly Detection

In Chapter 4, we introduced a graph encoder-decoder model for unsupervised anomaly detection. We also proposed a novel pooling strategy that utilizes locality-constrained linear coding for feature encoding. This pooling mechanism involves solving a least-squares optimization problem with a locality regularization term to obtain a cluster assignment matrix. By considering locality, our

pooling operation reduces the impact of irrelevant information present in the global graph structure, leading to more robust and representative feature extraction, which is essential for accurately identifying anomalies and distinguishing them from normal graph nodes. In the encoding stage of our model architecture, we used a multi-layer graph convolutional network encoder, followed by the pooling operation. In the decoding, we employed an unpooling operation, followed by a graph deconvolutional network decoder to decode the graph-structured data. Through our experimental evaluations, we demonstrate that our model, which incorporates the proposed pooling and unpooling layers in conjunction with locality-constrained linear coding, outperforms competing baselines on six benchmark datasets across a variety of evaluation metrics, showcasing its superiority in anomaly detection tasks.

5.2 Limitations

While the proposed node classification model demonstrates promising results for end-to-end semi-supervised learning on graphs and effectively addressing oversmoothing, its performance is contingent on the process of optimizing the hyperparameter related to anisotropic diffusion. This optimization is conducted through grid search and cross-validation for each dataset. Another drawback of the proposed model is its restriction to considering only immediate neighbors. This constraint can be overcome by adopting a higher-order message-passing approach, which leverages multi-hop neighbors by utilizing the powers of the adjacency matrix and hence aggregating learned node representations from both immediate and distant neighbors.

For the proposed semi-supervised anomaly detection framework, scalability becomes a primary concern, especially when applied to large graphs. As the graph size increases, so do the computational and memory requirements, potentially limiting its applicability in scenarios involving extensive real-world graphs. Moreover, the model's reliance on a minimal quantity of labeled data for proficient anomaly detection can be challenging in situations where acquiring such data is exceptionally difficult or costly.

In our unsupervised anomaly detection framework, the strength of the pooling approach in capturing local patterns may inadvertently lead to a vulnerability to anomalies that manifest as subtle, global deviations within the graph. This focus on local context may limit its effectiveness in detecting such anomalies. Furthermore, the encoder-decoder framework's dependency on coarsening and reconstructing the entire graph may pose scalability challenges, particularly when dealing with highly complex or large-scale graphs, potentially restricting its use in resource-constrained real-world applications.

5.3 Future Work

Several interesting research directions, motivated by this thesis, are discussed below:

5.3.1 Spatial-Temporal Graph Autoencoder for Anomaly Detection

Extensive research has been conducted on the topic of video anomaly detection over the years [150]. This interest has arisen due to the overwhelming volume of video content that exceeds our capacity for manual analysis. The human body skeleton can be modeled as a graph, where body joints represent nodes and bones represent edges. pose keypoints extracted from a video sequence are represented as a temporal sequence of pose graphs. We aim to apply spatial-temporal graph convolutional networks for detection of abnormal human actions. More specifically, we intend to apply a deep temporal graph autoencoder based architecture for embedding the temporal pose graphs while incorporating a pooling mechanism to cluster input data.

5.3.2 Graph Link Prediction for Anomaly Detection

Detecting anomalies patterns within graphs is a critical task with numerous practical applications, such as fraud detection and network security. We plan to explore the use of GCN-based models for graph link prediction, with a specific focus on enhancing anomaly detection through the integration of skip connections. To mitigate the over-smoothing problem that can occur when applying multiple graph convolutions, we will integrate skip connections into our GCN architecture. The goal of our research is to develop a novel approach that leverages GCNs to predict anomalous graph links. This approach will enable us to learn a robust graph embeddings for each node, thus improving the overall predictive accuracy and anomaly detection capabilities.

References

[1] T. Kipf and M. Welling, "Semi supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, pp. 1–14, 2017.

[2] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proc. SIGGRAPH*, pp. 317–324, 1999.

[3] Y. Zhou, H. Zheng, X. Huang, S. Hao, D. Li, and J. Zhao, "Graph neural networks: Taxonomy, advances, and trends," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 1, pp. 1–54, 2022.

[4] M. Mesgaran and A. Ben Hamza, "Anisotropic graph convolutional network for semi-supervised learning," *IEEE Transactions on Multimedia*, vol. 23, pp. 3931–3942, 2021.

[5] M. Mesgaran and A. Ben Hamza, "Graph fairing convolutional networks for anomaly detection," *Pattern Recognition*, 2023.

[6] M. Mesgaran and A. Ben Hamza, "A graph encoder-decoder network for unsupervised anomaly detection," *Neural Computing and Applications*, 2023.

[7] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proc. International Conference on Machine Learning*, pp. 1168–1175, 2008.

[8] Z. Yang, W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. International Conference on Machine Learning*, pp. 40–48, 2016.

[9] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[10] L. Lu, Y. Lu, R. Yu, H. Di, L. Zhang, and S. Wang, "GAIM: Graph attention interaction model for collective activity recognition," *IEEE Transactions on Multimedia*, vol. 22, no. 2, pp. 524–539, 2020.

[11] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. Conference on Knowledge Discovery and Data Mining*, pp. 701–710, 2014.

[12] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. Conference on Knowledge Discovery and Data Mining*, pp. 855–864, 2016.

[13] W. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin*, 2017.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Advances in Neural Information Processing*, pp. 3111–3119, 2013.

[15] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *International Conference on Learning Representations*, 2016.

[16] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2018.

[17] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

[18] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," pp. 5425–5434, 2017.

[19] R. Liao, Z. Zhao, R. Urtasun, and R. Zemel, "LanczosNet: Multi-scale deep graph convolutional networks," in *International Conference on Learning Representations*, 2019.

[20] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph wavelet neural network," in *International Conference on Learning Representations*, 2019.

[21] F. Wu, T. Zhang, A. de Souza Jr., C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. International Conference on Machine Learning*, 2019.

[22] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. International Conference on Machine Learning*, 2018.

[23] L. Zhao and L. Akoglu, "PairNorm: Tackling oversmoothing in GNNs," in *International Conference on Learning Representations*, 2020.

[24] D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, vol. 54, no. 1, pp. 45–66, 2004.

[25] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: A review," *ACM Computing Surveys*, vol. 54, no. 2, pp. 1–38, 2021.

[26] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proc. International Conference on Machine Learning*, pp. 4393–4402, 2018.

[27] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," in *Proc. International Conference on Learning Representations*, 2019.

[28] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *Proc. SIAM International Conference on Data Mining*, pp. 594–602, 2019.

[29] A. Kumagai, T. Iwata, and Y. Fujiwara, "Semi-supervised anomaly detection on attributed graphs," *arXiv preprint arXiv:2002.12011*, 2020.

[30] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Combining neural networks with personalized pagerank for classification on graphs," in *Proc. International Conference on Learning Representations*, 2019.

[31] G. Li, M. Muller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?," in *Proc. IEEE International Conference on Computer Vision*, pp. 9267–9276, 2019.

[32] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. International Conference on Machine Learning*, pp. 1725–1735, 2020.

[33] X. Wang, B. Jin, Y. Du, P. Cui, Y. Tan, and Y. Yang, "One-class graph neural networks for anomaly detection in attributed networks," *Neural Computing and Applications*, vol. 33, no. 18, pp. 12073–12085, 2021.

[34] S. Zhou, Q. Tan, Z. Xu, X. Huang, and F.-l. Chung, "Subtractive aggregation for attributed network anomaly detection," in *Proc. ACM International Conference on Information & Knowledge Management*, pp. 3672–3676, 2021.

[35] Y. Pei, T. Huang, W. van Ipenburg, and M. Pechenizkiy, "ResGCN: attention-based deep residual modeling for anomaly detection on attributed networks," *Machine Learning*, vol. 111, no. 2, pp. 519–541, 2022.

[36] Z. Zhuang, K. M. Ting, G. Pang, and S. Song, "Subgraph centralization: A necessary step for graph anomaly detection," in *Proc. SIAM International Conference on Data Mining*, 2023.

[37] J. Duan, S. Wang, P. Zhang, E. Zhu, J. Hu, H. Jin, Y. Liu, and Z. Dong, "Graph anomaly detection via multi-scale contrastive learning networks with augmented view," in *Proc. AAAI Conference on Artificial Intelligence*, 2023.

[38] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gomez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, 2015.

[39] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-toend deep learning architecture for graph classification," in *Proc. AAAI Conference on Artificial Intelligence*, pp. 4438–4445, 2018.

[40] X. Zheng, B. Zhou, J. Gao, Y. G. Wang, P. Lio, M. Li, and G. Montufar, "How framelets enhance graph neural networks," in *Proc. International Conference on Machine Learning*, 2021.

[41] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018.

[42] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. International Conference on Machine Learning*, vol. 97, pp. 3734–3743, 2019.

[43] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proc. International Conference on Machine Learning*, pp. 874–883, 2020.

[44] H. Gao and S. Ji, "Graph U-Nets," in *Proc. International Conference on Machine Learning*, pp. 2083–2092, 2019.

[45] H. Krim and A. Ben Hamza, *Geometric methods in signal and image analysis*. Cambridge University Press, 2015.

[46] S. Biasotti, A. Cerri, M. Abdelrahman, and et al., "SHREC'14 track: Retrieval and classification on textured 3D models," in *Proc. Eurographics Workshop on 3D Object Retrieval*, pp. 111–120, 2014.

[47] S. Biasotti, A. Cerri, M. Aono, and et al., "Retrieval and classification methods for textured 3D models: a comparative study," *The Visual Computer*, vol. 32, pp. 217–241, 2016.

[48] D. Pickup, X. Sun, P. L. Rosin, and et al., "Shape retrieval of non-rigid 3d human models," *International Journal of Computer Vision*, vol. 120, pp. 169–193, 2007.

[49] M. Masoumi and A. Ben Hamza, "Shape classification using spectral graph wavelets," *Applied Intelligence*, vol. 47, pp. 1256–1269, 2017.

[50] Y. Zhang and A. Ben Hamza, "Vertex-based diffusion for 3-D mesh denoising," *IEEE Transactions on Image Processing*, vol. 16, pp. 1036–1045, 2007.

[51] E. E. Abdallah, A. Ben Hamza, and P. Bhattacharya, "Spectral graph-theoretic approach to 3D mesh watermarking," in *Proceedings of Graphics Interface*, pp. 327–334, 2007.

[52] E. E. Abdallah, A. Ben Hamza, and P. Bhattacharya, "Watermarking 3D models using spectral mesh compression," *Signal, Image and Video Processing*, vol. 3, pp. 375–389, 2009.

[53] P. Kazienko and T. Kajdanowicz, "Label-dependent node classification in the network," *Neurocomputing*, vol. 75, no. 1, pp. 199–209, 2012.

[54] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proc. International Conference on World Wide Web*, pp. 1067–1077, 2015.

[55] L. Cai and S. Ji, "A multi-scale approach for graph link prediction," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3308–3315, 2020.

[56] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[57] C. Liu, Q. Zhong, X. Ao, L. Sun, W. Lin, J. Feng, Q. He, and J. Tang, "Fraud transactions detection via behavior tree with local intention calibration," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3035–3043, 2020.

[58] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[59] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. AAAI Conference on Artificial Intelligence*, pp. 3538–3545, 2018.

[60] J. Park, M. Lee, H. J. Chang, K. Lee, and J. Y. Choi, "Symmetric graph convolutional autoencoder for unsupervised graph representation learning," in *Proc. of the IEEE/CVF International Conference on Computer Vision*, pp. 6519–6528, 2019.

[61] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 723–731, 2019.

[62] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in *Proc. International Conference on Learning Representations*, 2020.

[63] T. Chen, K. Zhou, K. Duan, W. Zheng, P. Wang, X. Hu, and Z. Wang, "Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 2769–2781, 2023.

[64] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?," in *Proc. IEEE/CVF International Conference on Computer Vision*, pp. 9267–9276, 2019.

[65] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proc. International Conference on Learning Representations (ICLR)*, 2019.

[66] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. International Conference on Machine Learning*, pp. 5453–5462, 2018.

[67] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 338–348, 2020.

[68] A. Ortega, P. Frossard, J. Kovacevic, J. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[69] S. Bhagat, G. Cormode, and S. Muthukrishnan, *Node Classification in Social Networks*. In: Aggarwal C. (eds), Springer, 2011.

[70] F. Huang, X. Li, S. Zhang, J. Zhang, J. Chen, and Z. Zhai, "Overlapping community detection for multimedia social networks," *IEEE Transactions on Multimedia*, vol. 19, no. 8, pp. 1881–1893, 2017.

[71] L. Xu, T. Bao, L. Zhu, and Y. Zhang, "Trust-based privacy-preserving photo sharing in online social networks," *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 591–602, 2019.

[72] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI Conference on Artificial Intelligence*, pp. 922–929, 2013.

[73] S. Wang, Z. Chen, X. Yu, D. Li, J. Ni, L. Tang, J. Gui, Z. Li, H. Chen, and P. Yu, "Heterogeneous graph matching networks for unknown malware detection," in *Proc. International Joint Conference on Artificial Intelligence*, pp. 3762–3770, 2013.

[74] S. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert, "Metric learning with spectral graph convolutions on brain connectivity networks," *NeuroImage*, vol. 169, no. 0, pp. 431–442, 2018.

[75] Y. Chen, J. Wang, Y. Bai, G. C. nón, and V. Saligrama, "Probabilistic semantic retrieval for surveillance videos with activity graphs," *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 704–716, 2019.

[76] Q. Peng and Y.-M. Cheung, "Automatic video object segmentation based on visual and motion saliency," *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3083–3094, 2019.

[77] J. Gao and C. Xu, "CI-GNN: Building a category-instance graph for zero-shot video classification," *IEEE Transactions on Multimedia*, 2020.

[78] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proc. World Wide Web Conference*, 2018.

[79] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Conference on Knowledge Discovery and Data Mining*, 2018.

[80] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Advances in Neural Information Processing*, pp. 1–10, 2018.

[81] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing*, pp. 1024–1034, 2017.

[82] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[83] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2019.

[84] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI Conference on Artificial Intelligence*, pp. 3538–3545, 2018.

[85] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *International Conference on Learning Representations*, 2019.

[86] J. Weickert, *Anisotropic Diffusion in Image Processing*. ECMI Series, Teubner-Verlag, 1998.

[87] M. Black, G. Sapiro, D. Marimont, and D. Heeger, "Robust anisotropic diffusion," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 421–432, 1998.

[88] Y. Zhang and A. B. Hamza, "Vertex-based diffusion for 3-D mesh denoising," *IEEE Transactions on Image Processing*, vol. 16, no. 4, pp. 1036–1045, 2007.

[89] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.

[90] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*, pp. 639–655, 2012.

[91] X. Wu, Z. Li, A. So, J. Wright, and S. f. Chang, "Learning with partially absorbing random walks," in *Advances in Neural Information Processing*, pp. 3077–3085, 2012.

[92] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, , and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[93] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. International Conference on Machine Learning*, pp. 40–48, 2016.

[94] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[95] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, pp. 2579–2605, 2008.

[96] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, 2009.

[97] K. Doshi and Y. Yilmaz, "Online anomaly detection in surveillance videos with asymptotic bound on false alarm rates," *Pattern Recognition*, vol. 114, 2021.

[98] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, , and R. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[99] F. Zhang, S. Kan, D. Zhang, Y. Cen, L. Zhang, and V. Mladenovic, "A graph model-based multiscale feature fitting method for unsupervised anomaly detection," *Pattern Recognition*, vol. 138, 2023.

[100] L. A. S. Arias, C. W. Oosterlee, and P. Cirillo, "AIDA: Analytic isolation and distance-based anomaly detection algorithm," *Pattern Recognition*, vol. 141, 2023.

[101] R. Wang, K. Nie, T. Wang, Y. Yang, and B. Long, "Deep learning for anomaly detection," in *Proc. International Conference on Web Search and Data Mining*, pp. 894–896, 2020.

[102] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[103] H. Cevikalp, B. Uzun, Y. Salk, H. Saribas, and O. Köpüklü, "From anomaly detection to open set recognition: Bridging the gap," *Pattern Recognition*, vol. 138, 2023.

[104] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," in *Proc. International Conference on Learning Representations*, 2017.

[105] T. Schlegl, P. Seeböck, S. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks," *Medical Image Analysis*, vol. 54, pp. 30–44, 2019.

[106] F. D. Mattia, P. Galeone, M. D. Simoni, and E. Ghelfi, "A survey on GANs for anomaly detection," *arXiv preprint arXiv:1906.11632*, 2019.

[107] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan, "Do deep generative models know what they don't know?," in *Proc. International Conference on Learning Representations*, 2019.

[108] C. Huang, M. Li, F. Cao, H. Fujita, Z. Li, and X. Wu, "Are graph convolutional networks with random weights feasible?," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[109] G. Taubin, "A signal processing approach to fair surface design," in *Proc. SIGGRAPH*, pp. 351–358, 1995.

[110] G. Taubin, T. Zhang, and G. Golub, "Optimal surface smoothing as filter design," in *Proc. European Conference on Computer Vision*, 1996.

[111] D. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[112] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2018.

[113] F. M. Bianchi, D. Grattarola, C. Alippi, and L. Livi, "Graph neural networks with convolutional ARMA filters," *arXiv preprint arXiv:1901.01343*, 2019.

[114] A. Wijesinghe and Q. Wang, "DFNets: Spectral CNNs for graphs with feedback-looped filters," in *Advances in Neural Information Processing Systems*, 2019.

[115] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. International Conference on Learning Representations*, 2015.

[116] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[117] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[118] J. Wu, J. He, and Y. Liu, "ImVerde: Vertex-diminished random walk for learning network representation from imbalanced data," in *Proc. IEEE International Conference on Big Data*, pp. 871–880, 2018.

[119] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *Advances in Neural Information Processing Systems*, 2020.

[120] J. Lee, Y. Oh, Y. In, N. Lee, D. Hyun, and C. Park, "GraFN: Semi-supervised node classification on graph with few labels via non-parametric distribution assignment," in *Proc. SIGIR Conference on Research and Development in Information Retrieval*, 2022.

[121] D. Chen, Y. Lin, G. Zhao, X. Ren, P. Li, J. Zhou, and X. Sun, "Topology-imbalance learning for semi-supervised node classification," in *Advances in Neural Information Processing Systems*, 2021.

[122] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," *The Journal of Open Source Software*, 2018.

[123] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[124] K. Xu, M. Zhang, S. Jegelka, and K. Kawaguchi, "Optimization of graph neural networks: Implicit acceleration by skip connections and more depth," in *Proc. International Conference on Machine Learning*, 2021.

[125] G. Pang, A. van den Hengel, C. Shen, and L. Cao, "Toward deep supervised anomaly detection: Reinforcement learning from partially labeled anomaly data," in *Proc. ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1298–1308, 2021.

[126] F. Liu, X. Ma, J. Wu, J. Yang, S. Xue, A. Beheshti, C. Zhou, H. Peng, Q. Z. Sheng, and C. C. Aggarwal, "DAGAD: Data augmentation for graph anomaly detection," in *Proc. IEEE International Conference on Data Mining*, pp. 259–268, 2022.

[127] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 131, 2021.

[128] H. Tao, J. Qiu, Y. Chen, V. Stojanovic, and L. Cheng, "Unsupervised cross-domain rolling bearing fault diagnosis based on time-frequency information fusion," *Journal of the Franklin Institute*, vol. 360, no. 2, pp. 1454–1477, 2023.

[129] X. Song, P. Sun, S. Song, and V. Stojanovic, "Finite-time adaptive neural resilient dsc for fractional-order nonlinear large-scale systems against sensor-actuator faults," *Nonlinear Dynamics*, vol. 111, pp. 12181–12196, 2023.

[130] X. Song, C. Wu, V. Stojanovic, and S. Song, "1 bit encoding-decoding-based event-triggered fixed-time adaptive control for unmanned surface vehicle with guaranteed tracking performance," *Control Engineering Practice*, vol. 135, 2023.

[131] B. Perozzi and L. Akoglu, "Scalable anomaly ranking of attributed neighborhoods," in *Proc. SIAM International Conference on Data Mining*, pp. 207–215, 2016.

[132] J. Li, H. Dani, X. Hu, and H. Liu, "Radar: Residual analysis for anomaly detection in attributed networks," in *Proc. International Joint Conference on Artificial Intelligence*, pp. 2152–2158, 2017.

[133] Z. Peng, M. Luo, J. Li, H. Liu, and Q. Zheng, "ANOMALOUS: A joint modeling approach for anomaly detection on attributed networks," in *Proc. International Joint Conference on Artificial Intelligence*, pp. 3513–3519, 2018.

[134] J. Tang, J. Li, Z. Gao, and J. Li, "Rethinking graph neural networks for anomaly detection," in *Proc. International Conference on Machine Learning*, vol. 162, pp. 21076–21089, 2022.

[135] E. Ranjan, S. Sanyal, and P. Talukdar, "ASAP: Adaptive structure aware pooling for learning hierarchical graph representations," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5470–5477, 2020.

[136] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong, "Locality-constrained linear coding for image classification," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3360–3367, 2010.

[137] J. Li, J. Li, Y. Liu, J. Yu, Y. Li, and H. Cheng, "Deconvolutional networks on graph data," in *Advances in Neural Information Processing Systems*, pp. 21019–21030, 2021.

[138] C. Li and A. Ben Hamza, "A multiresolution descriptor for deformable 3D shape retrieval," *The Visual Computer*, vol. 29, pp. 513–524, 2013.

[139] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Learning structural node embeddings via diffusion wavelets," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1320–1329, 2018.

[140] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 990–998, 2008.

[141] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 817–826, 2009.

[142] Y. Liu, Z. Li, S. Pan, C. Gong, C. Zhou, and G. Karypis, "Anomaly detection on attributed networks via contrastive self-supervised learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[143] K. Ding, J. Li, and H. Liu, "Interactive anomaly detection on attributed networks," in *Proc. ACM International Conference on Web Search and Data Mining*, pp. 357–365, 2019.

[144] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 93–104, 2000.

[145] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "SCAN: a structural clustering algorithm for networks," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 824–833, 2007.

[146] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *International Conference on Learning Representations*, 2019.

[147] K. Ding, Q. Zhou, H. Tong, and H. Liu, "Few-shot network anomaly detection via cross-network meta-learning," in *Proc. ACM Web Conference*, pp. 2448–2456, 2021.

[148] Y. Chen, W. Luo, Y. Hao, and H. Jiang, "Anomaly detection of distribution network based on adversarial dual autoencoder," in *Proc. Journal of Physics: Conference Series*, vol. 2384, 2022.

[149] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[150] A. Markovitz, G. Sharir, I. Friedman, L. Zelnik-Manor, and S. Avidan, “Graph embedded pose clustering for anomaly detection,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10539–10547, 2020.